BUDS: THE BERKELEY URBAN DATA SYSTEM

by

P. P. Macri

Memorandum No. ERL-M412

9 November 1983

BUDS: The Berkeley Urban Data System

by

Philip Pasquale Macri

Memorandum No. ERL-M412

9 November 1973

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

BUDS: The Berkeley Urban Data System

Ph.d. Philip P. Macri Dept. of Electrical Engineering and Computer Sciences

Chairman of Committee

ABSTRACT

One of the difficulties encountered by the urban analyst or planner is the fact that he is dealing with a large and complex system. Because of this experimentation is almost always impossible leaving observation and study as primary modes of analysis. However empirical studies are impeded by the enormous volume and nature of urban data. One of the things sorely needed is a capability to manipulate large quantities of urban data and at the same time present it in a form which would aid comprehension. The Berkeley Urban Data System (BUDS) provides this capability.

Aside from its large volume urban data has other properties which must be dealt with if the data is to be used effectively. First of all it is geographic in nature. In many instances the analyst is not only interested in data for a certain area but in neighboring areas as well. For example the distribution of blacks and office buildings is not uniform over a census tract map of San Francisco. Relative sizes, location and composition of areas are extremely important properties. Topographical properties are also important. For example the existence of a deepwater port will greatly influence among other things the type of economic activities and population distribution in its vicinity. Data comprehension is enhanced by pictorial presentations such as shaded maps. The map portion conveys geographic and topographic properties while shading

conveys quantitative information.

Geographical boundary definitions change from tally to tally. For example, the nine county Bay Area Census of Population and Housing for 1970 has approximately 28% more census tracts than the 1960 census. So a comparison of 1960 and 1970 data at the census tract level cannot be made on the same map.

Since different agencies collect data, and do it for different purposes it is obvious that they rarely use the same map to define tally areas. However, it is sometimes possible to manipulate data collected over one map to conform to other maps. For example, census tract data can be aggregated up to county level.

Data is also time dependent. Not only do geographical boundaries change but population shifts, economic activities change, etc. Since the dynamic evolution of urban systems is a primary concern of the analyst and planner intertemporal studies are important. Therefore a locational correspondence must be made on data over time.

These properties were taken into account in the design and implementation of BUDS. BUDS, as described in this thesis is a portion of what is to be: a comprehensive, general interactive computer graphics urban research system. The primary system objective is that it be an effective teaching or research tool. Currently BUDS is implemented on a 16K-16 bit computer connected to an IBM 2250 graphics console.

The simplest description of BUDS is as follows: a partitioned map-as opposed to a road map-is displayed on a 12 x 12 inch cathode-ray tube upon which a preselected variable is displayed in the form of dot shading whose density is proportional to the value of the

variable in each of the partitions. This form of data presentation is highly intelligible. Quantitative results are also available.

The system is extremely simple to operate, embodies many automatic facilities and requires no computer knowledge to operate. Modularity in both programming and data structure was maintained enabling system modifications at minimal cost. Data structures are simple and it is easy to add or delete data.

The design and implementation of BUDS resulted in an efficient and highly effective interactive research tool for urban studies.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Pravin P. Varaiya for his many helpful suggestions and his patience through the many distressing equipment failures. I also wish to thank Professor Michael Stonebraker for his helpful criticisms on the writing of this thesis.

Finally I wish to thank the National Science Foundation for grant NSF-GK-10656x2 which supported this research.

TABLE of CONTENTS

LIST of FIGURES

CHAPTER 0

THE NECESSITY AND CRITERIA FOR BUDS

0.1. Introduction

One of the difficulties encountered by the urban analyst or planner is the fact that he is dealing with a large system with an accompanying large number of variables. Because urban systems tend to be large and complicated experimentation is almost always impossible. This leaves the analyst with observation and study as primary modes of analysis. However there is the problem of comprehending the data if only because of its sheer size. One of the things sorely needed is a capability to manipulate large quantities of urban data and at the same time present it in a form which would aid comprehension.

Aside from its large volume urban data has other properties which must be dealt with if the data is to be used effectively. First of all it is geographic in nature. In many instances the analyst is not only interested in data for a certain area but in neighboring areas as well. For example the distribution of blacks and office buildings is not uniform over a census tract map of San Francisco. Relative sizes, location and composition of areas are extremely important properties. Topographical properties are also important. For example, the existence of a deepwater port will greatly influence among other things the type of economic activities and population distribution in its vicinity. Data comprehension is enhanced by pictorial presentations such as shaded maps. The map portion conveys geographic and topographic properties while shading conveys quantitative information.

Generally, the investigator must rely on data gathered by a number of different data collection agencies, or particular studies. This implies non-uniformity of data which can occur in many ways even within agencies. First of all, the data accuracy may vary. Is this random sample data truly a random sample? That is, does it represent a true cross-section? Is the 25% sample as good as the 100% sample? Questions of this sort are beyond the scope of this thesis, and will not be discussed. Secondly, assuming the data to be accurate, there are problems when one wishes to use it. Geographical boundary definitions change from tally to tally. For example, the nine county Bay Area Census of Population and Housing for 1970 has approximately 28% more census tracts than the 1960 census. So a comparison of 1960 and 1970 data at the census tract level cannot be made on the same map.

Since different agencies collect data, and do it for different purposes, it is obvious that they rarely use the same map to define tally areas. However, it is sometimes possible to manipulate data collected over one map to conform to other maps. For example, census tract data can be aggregated up to county level.

Data is also time dependent. Not only do geographical boundaries change but population shifts, economic activities change, etc. Therefore if intertemporal studies are to be undertaken a locational correspondence must be made on the data over time.

For these reasons and others it was decided that computer manipulation of data would be the only feasible approach. A survey of existing data systems[1] was made and all were rejected for one or more reasons. If a system were to be adopted it would have to be

[1]The systems described in 7,8,9,12,13,14 →20 are typical of the ones surveyed. Others may be found in 21,22,23.

implemented on existing university computer equipment. Batch data manipulating system were ruled out because of inflexibility, inefficiency or they just didnot perform the tasks we had in mind. The interactive systems require large time-sharing machines which are not at our disposal and rental costs would be prohibitive. The idea of adopting pieces of different systems and piecing them together was deemed infeasible. For these reasons it was decided that if we wanted an effective urban data system we would have to design and implement our own. It was also decided that a comprehensive, general interactive urban research system was too ambigous a task at that point. Secondly the design and implementation of a portion of the system would serve as a pilot project to test the feasibility of such a system on the computer graphics equipment we decided to use. This pilot project has been completed and is the subject of this thesis. The resulting system is called the Berkeley Urban Data System which will be referred to in the text as BUDS.

0.2. System Objectives

BUDS, as described herein, is a portion of what is to be a comprehensive, general interactive computer graphics urban research system. The overall system is envisioned to encompass two fully integrated portions; an information storage retrieval system and an analysis system. The primary system objective is that it be an effective teaching or research tool. Currently the system is implemented on a Digital Scientific Corportation Meta IV. It is a 16K-16 bit general purpose computer [4] which is microprogramable and emulating an IBM 1130. (Figure 1 illustrates the hardware.) It

has a single IBM 2310 type disk unit with a storage capacity of 512,000 words. This amount of secondary storage is a limiting factor and is one of the reasons why a general information retrieval system was not contemplated in the initial design. Funds have been appropriated and negotiations are underway to purchase a 2314-type disc unit.

A Documation M600 cardreader is the principle input device and an IBM selectric typewriter is the principle output device. A modified IBM 2250 model III Graphics Console operating on a cycle-steal basis from the Meta IV memory is the graphics portion of the system. The 2250 is a refresh vector graphics system with an alphanumeric keyboard, program function keyboard[2] and light pen. The light pen is a light sensitive device and is used to identify objects on the CRT. Although this equipment is expensive similar equipment is currently on the market at very reasonable prices. For example the Digital Equipment Corporation GT44 Graphics System with a PDP11/40 costs $34,500 and is comparable to our equipment in present usage with the exception of the graphics terminal.[3] The costs of this type of equipment continues to decrease with time making such systems attractive to urban planning and research agencies with limited financial resources.

The availability of the graphics equipment and the geographical nature of urban data prompted the idea of presenting urban data in

---

[2]The acronym PFKB will be used for program function keyboard.

[3]The screen is approximately 65% the size of the IBM 2250 and on a line basis it is estimated that 25% fewer lines may be drawn before flicker occurs. Aside from these drawbacks the GT44 has some advantages over the IBM 2250. These include programable intensity, character generating hardware, etc.

a manner as illustrated in Figs 7 and 8. The simplest description of BUDS is as follows: a partitioned map-as opposed to a road map-is displayed on a 12 x 12 inch cathode-ray tube upon which a preselected variable is displayed in the form of dot shading whose density is proportional to the value of the variable in each of the partitions.

Using this as an introduction we now focus attention on the system objectives which guided and hopefully will continue to guide the future development of BUDS.

1. Intelligibility of data presentation. By this we mean the following attributes characterize data display:

A. It presents or preserves the following locational data:

1) The physical shape of the area or areas in question are presented.

2) The relative size of the areas is preserved.

3) Bodies of water, rivers, major tributaries and transportation networks are displayed. By this we mean the relative location and distance of neighboring areas is displayed.

B. A visual comparison of an urban variable in the form of shading in neighboring areas is made. That is the density of the shading is proportional to the magnitude of the variable.

Intelligibility applies not only to queries results but to generation of queries as well. For example a directory of the variables stored in the computer data bank is presented to the user in a noncluttered highly intelligible form (Fig. 6). A simple mechanism allows him to scan through the directory. Selection of variables to be included in his query is accomplished by touching

that variable with the light pen. The query definition itself is also highly intellible (Fig. 8a).

2. It allows for a quantitative study as well as a qualitative study. That is, numerical results can also be obtained from a query.

3. The system is interactive and as such provides excellent response time.

4. The system is extremely simple to use and requires no programming or computer knowledge to operate.[4] Once activated program control is accomplished through the program function keyboard, PFKB, of the 2250 graphics console (Fig. 2a). The user simply selects the button corresponding to the desired function. Throughout program execution the user is guided as to which subset of functions are allowable or actions are expected at that particular moment. This is done by illuminating those button switches in the PFKB which correspond to set of allowable functions and by displaying messages on the CRT.

The system is "idiot proof." Automatic facilities insure that illegal actions are not taken. Thus the system is fully protected against the reachability of ambiguous or calamitous states.

5. Modularity:

Modularity in programming and data structures is maintained so that new functions may be added and old ones modified with minimal reprogramming of unmodified functions.

6. Data structures are simple and it is easy to add or delete data. Furthermore careful consideration was given to data structures so that it would be easy to design and implement a data management

---

[4] It takes approximately 25 minutes to instruct people to use the system.

program which would automatically make the appropriate changes to add or delete data.

7. BUDS provides the user with the capability of performing macro or micro studies[5] and to switch from one to the other easily. The system allows this without redundancy of data by automatically aggregating data for small areas so that it represents data for large areas. For example aggregating or summing census tract data to county level.

8. Automatic facilities are provided to guarantee query integrity. This occurs at two levels. First the query is automatically checked for syntactical errors. Secondly the query is checked for data consistency. Urban data has locational properties therefore it is important to ensure that only urban data over common locations are combined. For example forming an expression in which census tract data of San Francisco is combined with that of Los Angeles is not allowed.

9. Intertemporal studies can be easily performed. The concept of time in urban systems is of fundamental importance. It is the dynamic evolution of urban systems that is of primary concern to the analyst and planner. They are constantly seeking answers to questions of the sort: How does a city grow? Therefore a facility to perform intertemporal studies with ease is especially important. (Fig. 9)

0.3. Some Remarks on the Usage of BUDS

From the users point of view these objectives have interesting

[5] These approaches are described by Birch [5] and Rogers [10]. Macro studies analyze the urban region in terms of highly aggregated units while the latter focuses on disaggregated units.

implications. First of all the user gets a complete picture for the question he has asked. Relevant attributes such as geography and transportation facilities and the influence of neighboring areas are before him. Because the system is interactive and several variables (up to ten) can be selected beforehand, the user can sit before the console and get the whole history of the area in question rapidly. As an example, he can quickly spot population shifts from one time period to the next. Since pictures can be stored in secondary memory, they can be accessed rapidly (without reconstructing them) for comparisons.

The system can be used to validate hypothesis and theories quickly. Continuing with the preceding example, the investigator can check theories of population shifts by querying the system for the variables thought to cause population shifts. Indeed, the user may even generate a new hypothesis and test it immediately. The interactive nature, speed, and economies of BUDS afford the user latitudes that he normally would not have with conventional batch processing or manual means. Continuing the proceeding example: not only could the user spot population shifts but he can also ascertain the kind of shift, (young, minority, etc.) and accompanying changes (property tax, housing age, construction of new freeway); all within a few minutes.

From the preceding remarks it is obvious that BUDS was designed to carry on empirical studies such as the well known work of Hoover and Vernon [6] in Anatomy of a Metropolis. This important piece of work describes in empirical detail the trends and forces the authors observed to have been instrumental in the development of the New York

metropolitan area. The amount of effort that went into data manipulation and presentation for this research must have been monumental. BUDS has the capability of performing this kind of study orders of magnitudes faster.

Urban systems are not understood very well. In fact, there is precious little in the way of theory. Attempts to apply existing theory to real life situations have not been entirely successful [1,2]. One of the reasons for this state of being is the lack of a tool [3,5,11] with which empirical studies can be performed effectively and efficiently. It is hoped that BUDS will provide our Urban Systems Group with this capability and that it will enable us to better understand urban systems thus augmenting our theoretical studies.

From a computer science point of view BUDS can be viewed as an example of man-machine interaction. The design of BUDS involved at least four basic research areas.

1. Interactive computer systems
2. Design of data base structures and access methods
3. Data presentation technqiues
4. Human factors.

The interaction of these four areas as applied to the particular problem at hand resulted in a tool which enhances our urban research capabilities.

First of all the user is not forced to think or interpret data in a peculiar fashion. Furthermore system operation is also natural. That is the user needs no knowledge of computers or computer programming. He simply choses the appropriate sequence of functions necessary to accomplish his goal. Chapter 2 shows that selection of this sequence

of functions is virtually obvious in that the user simply executes his normal thought process. Thus the user is not distracted by operational procedures and can concentrate on the real problem at hand.

Secondly BUDS aids the analyst not only by guiding him but also in helping him to translate his query into a precise statement. For example in many instances the analyst's query is "fuzzy." A typical question is "Where do the young people live?" The term young is relative and must be translated into a specific age group. Even the term people may be translated into male, female, married, etc. The presentation of a directory of variables which is easily scanned can help the user formulate the precise queries he really wants. For example the above question can be translated into "What is the distribution of black males between the ages of 0-19 as a fraction of the total population in San Francisco in 1970?"

This introduction is concluded with a brief description of the contents of the remaining chapters.

Because of the attributes of urban data namely time, quantity and location considerable attention was given to their interrelations. This resulted in data structures which are efficient and flexible. Chapter I is concerned with basic data definitions which explicitly tie together locational data to quantitative data with respect to time.

Chapter II describes the functions which allow the user to generate and receive answers to queries.

Chapter III describes the data structures and how they are used.

Finally Chapter IV contains concluding remarks.

CHAPTER 1

URBAN DATA AND ITS TREATMENT IN BUDS

1.1 Introduction

This chapter illustrates the interconnection and types of locational data and quantitative data in BUDS. In so doing it also gives an overall view of how urban data is manipulated. It does this through a series of definitions. The formal structure of quantitative data files is given. Although all of the definitions presented are not used in this chapter they are presented here to explicitly define the nature of data treated in BUDS and for the sake of continuity. The reader may skip to Chapter II and return if detailed knowledge is desired.

1.2 Urban Data in BUDS

We now give a few definitions which will help to describe the treatment of urban data in BUDS. Only numerical data is treated. Practical experience indicates that this is not a serious restriction because when collected data of a qualitative nature is usually aggregated into a number. For example, the number of sound housing units in census tract BE001.

In any event, qualitative data can usually be quantized. As an example, the numbers 1,2, and 3 may be attached to the attributes sound, deteriorating and dilapidated housing units, respectively.

We also assume that the quantative data is taken over finite areas, and that it is distributed uniformly over these areas.

Aside from its name, there are three attributes which characterize an urban variable:

1. time

2. quantity or quality (assumed to be a real number)

3. location

As an example, the population (name of urban variable) of census tract BE001 (location or area over which tally took place) in 1960 (time) was 4050 (numeric describing quantity).

We begin with describing the treatment of location.

1.21 Map We consider a fixed, finite, two dimensional area A partitioned into a finite number of disjoint polygonal subsets $P_1,\ldots,P_n$. Hence

$$A = \bigcup_{i=1}^{m} P_i \text{ and } P_i \cap P_j = \phi \text{ if } i \neq j$$

A map M is any partition $\{S_1,\ldots,S_n\}$ of the area A which is a refinement or cover of the partition $\{P_1,\ldots,P_m\}$. That is for each $i, 1 \leq i \leq n$ there exists integers $i_1,\ldots,i_k$ such that

$$S_i = \bigcup_{j=1}^{k} P_{i_j}$$ furthermore

$$A = \bigcup_{i=1}^{n} S_i \text{ and } S_i \cap S_j = \phi \text{ for } i \neq j.$$

If $M = \{S_1,\ldots,S_n\}$, the subsets S_i are called sectors of M.

This definition is quite general since it allows sectors to be the union of any finite set of polygonal areas. Thus, sectors may be entirely within the outer boundaries of other sectors or they may consist of non contiguous polygonal areas, etc.

1.2.2. A map set is a collection of maps MS = $M_1,\ldots,M_q$. In what follows MS denotes the map set stored in BUDS.

1.2.3. Variable Name. A variable name, v_i, is a description of a variable. For example, total population is a variable name. Variable names belong to a finite set $V = \{v_1,\ldots,v_r\}$. Each data gathering effort such as the U.S. Census generates a subset of variable names V_ℓ. However, we restrict the members of V to those stored in BUDS. We shall see later that the set V serves as a directory for the retrieval of data in BUDS.

1.2.4 Data Element

Each data gathering effort generates a map, M_m, and a subset, V_ℓ, of variable names. Data is gathered for each sector $S_j \in M_m$ and each variable name $v_i \in V_\ell$, i.e. a numeric is assigned to each sector for each variable name. To insure uniformity data for several variables is usually gathered at the same time, t_k. For example, the 1970 U.S. Census of Population and Housing was completed over an interval of time which was small enough to rule out significant changes in the items being enumerated. Each numeric is called a data element. We now give a more formal definition.

Given a data gathering effort (census, inventory, etc.) which at time t_k generates a subset of variable names $V_\ell = \{v_1,\ldots,v_q\}$ and a map $M_m = \{S_1,\ldots,S_n\}$ a data element $DE_{ijk\ell m}$ is that quantity associated with the variable name $v_i \in V_\ell$ and the sector $S_j \in M_m$. Data elements may be thought of as representations of the functions

$$f_{\ell m}(v_i,S_j,t_k) = DE_{ijk\ell m}$$

for all $1 \leq i \leq q$ and all $1 \leq j \leq n$. As in previous instances the set of data elements is limited to those stored in BUDS.

We conclude this section with an example of a data element. The total population (v_i) in 1960 (t_k) in the MTC[1] zone 4 (S_j) was 13,900 according to the 1960 U.S. Census of Population.

1.2.5. Data Record. A data record, $DR_{ik\ell m}$, is a vector whose components are data elements. The components correspond to the sectors of the maps associated with the data elements. Using the notation of the previous section

$$DR_{ik\ell m} = (DE_{i1k\ell m}, \ldots, DE_{ink\ell m})$$

Data records as defined here are actual logical data records as implemented in BUDS.

1.2.6. Basic Variable Data File

A basic variable data file[2] $BVDF_m$ is an ordered set of data records. Continuing with the notation of sections 1.2.4 and 1.2.5

$$BVDF_m = (DR_{1k\ell m}, \ldots, DR_{rk\ell m})$$

with the following constraints. The data records in $BVDF_m$ must all be of the same dimension. This constraint is imposed only for data storage, retrieval and manipulation efficiency. It may be violated without any retrieval errors or reprogramming if the aforementioned

[1]The Metropolitan Transportation Commission, Claremont Hotel, Berkeley, California, a regional commission partitioned the 9 county Bay Area into 290 zones which are aggregations of census tracts.

[2]For the sake of brevity in the text the term data file will be used for basic variable data file whenever there is no possibility of confusion.

inefficiencies are tolerable. The constraint means that all data records in a particular data file must be over the same map or maps with the same number of sectors. This is a natural organization scheme which simplifies data management but is is not necessary.

The order of the data records is immaterial although it is advantageous to have certain natural groupings. For example, the records corresponding to the total, white, black and Spanish surname population should be sequential in the data file. Sections 3.2.1 and 3.2.2 will show that certain groupings of data records make the directory[3] more comprehensible and offer economies in data record address calculation.

This section is concluded with an example. The U.S. Census of Population and Housing for 1970, t_k, induced a subset of variable names $V_\ell = \{v_1,\ldots,v_\ell\}$ and a census tract map $M_m = \{S_1,\ldots,S_n\}$ for the nine county San Francisco Bay Area. With these, the Census generated a set of data records which form or are part of one or more data files. The variable names serve as pointers[4] to data records. A one-to-one correspondence between the components of a data record and the sectors of the map provides the mechanism through which data for individual census tracts may be accessed.

1.3. Transformation Mapping One Data Record Into Another.

Sometimes it is necessary to compare or examine urban data

---

[3] A directory of variable definitions stored in BUDS is presented to the user during query generation for light pen solution. Figure 6 illustrates a page of the directory.

[4] The mechanisms through which this is accomplished is discussed in sections 3.2.1 through 3.2.4.

defined over maps. As an example, suppose that we have census tract level data and we wish to look at the data at county level, or perhaps we wish to form a census tract per city type user generated variable.[5] For example the black population in each census tract of San Francisco divided by the total population of San Francisco. In this case we would have to aggregate (sum), the census tract data up to the city level. Note that it is not always possible to aggregate data defined over a lower level map to a higher level map. For example, the Department of Human Resources, State of California Yearly Employment Statistics collects data over local areas. There are seventeen (17) local areas in the nine county San Francisco Bay Area. These 17 areas comprise only a small fraction of the entire nine county Bay Area. Secondly, the Palo Alto area includes parts of San Mateo and Santa Clara County. In this case one could calculate the fractional part that lies in one county and the fractional part lying in the other county and form some kind of weighted average. However, it is not clear how this average should be weighted, especially since the data is taken over certain urban centers. The extrapolation to the rural parts of Santa Clara county, etc. would be difficult to make. In any event, extrapolating data in this fashion has been ruled out at present. Note however that it is possible to aggregate census tract data up to county level.

Formally we have:

A Data Record $DR_{ik\ell m} \in BVDF_m$ can be transformed into another Data Record $DR_{ik\ell p}$ where

[5] User generated or created variables are discussed in section 2.2.

$$M_m = \{S_1, S_2, \ldots, S_n\}$$

$$M_p = \{S_1, S_2, \ldots, S_r\}$$

and

$$0 < r \leq n$$

if for every $S_j \in M_m$ there exists a non empty sequence $\{a_{1j}, a_{2j}, \ldots, a_{fj}\}$ such that

$$S_j = \bigcup_{i=1}^{f} S_{a_{ij}}$$

Using this formulation is is easily seen that transformation from M_m to M_p defines an rxn matrix X[6] such that

$$M_p = X\, M_m$$

This obviously can be carried out using Data Files,

$$BVDF_p = X\, BVDF_m$$

but in implementation only data records need transformation.

This transformation eliminates considerable amounts of redundancy in the data at the cost of increased computation. It is through this transformation that the macro and micro approaches discussed in section 0.2 can be made. Implementation is discussed in Chapter 3.

These definitions are simple yet broad and they form the basis for the interconnection of maps and urban data. They also form the

[6] The components of this matrix are either 0 or 1 and the number of ones in any column is at most one.

basis for data storage structures (data bases) and retrieval mechanisms in BUDS which are discussed in Chapter 3.

CHAPTER 2

OPERATING BUDS

2.1. System Operation

The user activates BUDS by requesting the computer to execute DQEP, VARDP[1] or DDQEP. DDQEP will reinitialize the state of BUDS to a default value of time, T_o, whereas either of the other two programs will set the state of BUDS to what is was at the time BUDS was last stopped. From time to time the computer malfunctions putting BUDS in a nonrecoverable state. These states are nonreachable while operating BUDS and usually are nonsensical. When this occurs DDQEP must be executed.

Once BUDS is activated, program control is through the program function keyboard,[2] PFKB, of the IBM 2250 Graphics Console which is shown in Fig. 2a. The set of allowable functions, (Fig. 2a) can be partitioned into two classes of functions, the set of variable definition and editing functions and the set of map manipulation and display functions.[3] Throughout program execution the user is guided as to which subset of functions are allowable at that particular moment. This is done by illuminating the button switches in the PFKB, corresponding to each function in the allowable subset of functions. The selection of a non-allowable function is ignored.

---

[1] DQEP is the acronym for display queue editing program while VARDP is the acronym for variable display program.

[2] In what follows we will use the abbreviation PFKB for Program function keyboard.

[3] The former are in DQEP and the latter are in VARDP

2.2. Variable Definition and Editing Functions

This portion of the system enables the user to generate queries. As such it has an extensive editing facility to enable the user to generate variables using basic variables[4] and to modify previously user-generated variables. Each user-generated variable or created variable[5] is an arithmetic expression specified by the user; the arithmetic expression may involve up to ten basic variables. We call this expression the variable expression. Figure 3 shows a user generated variable with two basic variables whose expression indicates the user is interested in their difference. Functions 15 through 26 on the PFKB (Fig. 2) are dedicated to editing. Many automatic features are incorporated into these functions which aid the user. They also guarantee that the created variable makes sense (syntax and data consistency) and prevent the computer from getting into an ambiguous or calamitous state.

Up to ten created variables may be defined at any instant of time. Created variables are said to reside on the display queue and are referenced by queue numbers one through ten. Due to the limited screen size only three created variables may be displayed (Fig. 9) at one time during variable definition and editing. Their corresponding screen location are 1, 2 and 3 with 1 at the bottom. These screen location numbers are not to be confused with queue numbers which refer to variable position on the display queue.

---

[4]Basic variables are virtually synonymous with variable names defined in Section 1.2.3. Basic variables contain more information than variables names. In addition to discribing the variable (e.g. total population they also contain time (e.g. 1960) and locational information (e.g. county). See Section 2.2.3.

[5]From this point on the terms user-generated variable and created variable will be used synonymously. Furthermore the term variable will be used interchangeably with created variable or basic variable when there is no possibility of confusion.

Before discussing DQEP functions a few remarks concerning the method of creating variables in BUDS are in order. A great deal of emphasis was given to system objectives because the usefulness of the query and query system depends on how well queries represents what the analyst wants. To this end the system objectives dictate that BUDS must help the analyst to formulate queries. For this reason it was decided to allow the user to select basic variables from a directory displayed on the CRT and to express his query with a mathematical expression involving these basic variables. Figure 6 illustrates a page of the directory and Fig. 3 illustrates a created variable or query.

The ability to flip through pages of the directory displayed on the CRT and to choose basic variables with a light pen is superior to the method of query generation on many other systems. For example the IBM system [12,13] requires the user to type in the complete query. The IBM system is cumbersome in that the user must thumb through a paper bound directory for the proper basic variable and correctly type it on an alphanumeric keyboard. While doing this he must separate basic variables with operators. The resulting query aside from being difficult to generate is also difficult to read. In BUDS query specification is separated from basic variable selection. Basic variable selection is facilitated as well as query specification and comprehension. To specify the query letters corresponding to basic variables along with operators are typed in. The resulting query is more concise and easier to understand than those generated on the IBM system.

The next thirteen sections describe the functions associated with

creating and editing variables. Figure 2b illustrates the order in which these functions are accessible.

2.2.1. Creating a Variable

The user begins creation of a variable by selecting the START NEW VARIABLE function on the PFKB (function number 23 in Fig. 2a). If all ten queue numbers have been allocated, a warning is sounded[6] and the function is aborted. If all ten queue numbers have not been allocated, the message SCREEN LOCATION {0}[7] is displayed in the lower right corner of the screen. The user must then enter a screen location using the IBM 2250 alphanumeric keyboard (keyboard).[8]

Entering any number other than 1 through 3 aborts the function with a bleep. Entering a proper screen location number results in the following sequence of events:

If a variable is being displayed on the screen at the desired location its definition is stored on the disc and it is removed from the screen. The next position on the display queue is assigned to the new variable and the variable header is generated (Fig. 4a). The variable header contains information which controls the type of display and the queue number[9] of this variable. It is discussed in

---

[6] Error which cause a function abort are signaled by an audio signal which sounds like a "bleep." From now on this signal will be called a bleep.

[7] In what follows all characters within brackets {} in a message displayed on the screen may or must be replaced by the user using the alphanumeric keyboard. The brackets do not appear on the screen but merely serve the purpose of delineating certain characters for convenience in this text. The term within brackets usually is a default value and is replaced with characters as they are typed in by the user. Refer to section A.1 for details.

[8] In what follows the term keyboard will mean the IBM 2250 alphanumeric keyboard shown in Fig. 1.

[9] Queue numbers are allocated sequentially to create variables. Thus for example if seven variables have been created already, eight would be the next available queue number.

section 2.2.4.

2.2.2. The Append Function

By selecting the APPEND function (number 19 in Fig. 2a) the user is able to append basic variables to a variable. If there are no variables being displayed on the screen the function aborts with a bleep. Otherwise the user responds to the message SELECT APPENDING POSITION by selecting the exact location for appending a basic variable using the light pen. The function is aborted with a bleep if the user selects a non-variable item such as the message or if he selects an appending position in a variable which already has its full complement of basic variables. The appending position is the position just beneath the entity touched by the light pen.

The entity touched by the light pen is brightened to indicate this position. The user must now confirm or reject his selection using the REJECT or the CONTINUE/ACCEPT buttons on the PFKB. Rejection aborts the APPEND function. If the user accepts the appending position the message LIGHT PEN OR BASIC VARIABLE LIST appears, the functions RETURN and BASIC VARIABLE LIST (functions 30 and 17 of the PFKB respectively) are available for selection and the light pen is activated. Up to n basic variable may be selected, where n equals ten minus the number of basic variables already in the created variable.

The user may select basic variables appearing in any created variable on the screen with the light pen. If a basic variable is selected it is brightened and the selection must either be accepted or rejected using those functions on the PFKB. In the event basic variable selection is desired from the directory the function BASIC VARIABLE LIST (number 17 or the PFKB) is selected. Basic variable

selection may then be continued from the directory.

2.2.3. Selecting a Basic Variable from the Directory

Because the screen has a finite size the description of all variables stored in the data bank cannot be adequately displayed on the screen at once. Taking this into consideration and all of the system objectives, namely intelligibility, ease of use, ease of data insertion and deletion, and the fact that we are dealing with a small machine, the following strategy was adopted for the directory.

A number of year list pages are stored in secondary storage. Each year list page[10] consists of a list of years. For example years 1950, 1960 and 1970 may comprise a year list page. Each year on each page serves as a pointer to set of basic variables called a page of basic variables. If the number of basic variables for a particular year is such that their description cannot be adequately displayed on the screen at once, then that year is put onto the second year list page as well. The process is continued until all of the basic variables for that year are accounted for. Each year on a year list page corresponds to one and only one page of basic variables. Only one page of basic variables is displayed on the screen at one time. Basic variable selection is made from a basic variable page.

Having selected the BASIC VARIABLE LIST function a year list page[11] together with its page number is displayed on the screen.

---

[10] This structure is detailed at length in section 3.2.1.

[11] Presently there is only one year list page with only the years 1960 and 1970. This is due to the limited capacity of secondary storage and the available data.

The user must then indicate the page of basic variables from which he wishes to select a basic variable. This is accomplished by selecting a year from the year list page displayed on the CRT using the light pen. One of the years in the list is brighter than the rest of the entries on the list and is used as a default selection. In making his selection the user may be satisfied with the default year, select a different year from this page using the light pen or display another year list page by selecting the DISPLAY YEAR LIST function (number 16 in Fig. 2a). The message NEW PAGE NUMBER {0} is displayed in the lower right corner of the CRT and the desired page number is entered via the keyboard.[12]

Satisfied with his year selection the user displays the page of basic variables by selecting the BASIC VARIABLE LIST function (number 17 in Fig. 2a). This causes the list of basic variables corresponding to the above selection to be displayed (Fig. 6). At the bottom of the screen the page number and year indexing the page of basic variables is displayed.

In accordance with the system objectives advantage was taken of the peculiarities of urban data. Typically an urban variable has several attributes which were divided into two classes; primary attributes and secondary attributes.

A basic variable consists of a name[13] and primary and secondary attributes which characterize it and a map which determines the map aggregation level. For each urban variable there is only one list of primary attributes, one list of secondary attributes and one map

---

[12] If the user inputs the same page number as is currently displayed the function is aborted. If the user requests a non-existent year list page the function is aborted with a bleep. In either case the message disappears once page number input is terminated.

[13] If there are no primary modifiers then the name itself comprises the list of primary attributes and we say the name is null.

list in BUDS. Primary attributes modify the name so that the name need not be repeated. This is indicated in Fig. 6. The list of secondary attributes[14] modify the list of primary attributes providing a further description of the variable. These are typically parameters relating to race, sex, age, etc. They are included as a list so that the name and primary attributes need not be repeated for each member of the secondary attributes list.

Selecting a primary attribute with the light pen causes the appropriate secondary attribute list and map list to be displayed. Members of these lists once displayed are available for selection using the light pen. Thus a basic variable is specified by selecting with the light pen[15] one primary attribute, one secondary attribute from a list of secondary attributes if it exists and a member of the map list. The light pen selection of the basic variable may be accepted for inclusion in the created variable only by selecting the ACCEPT function on the PFKB.

The map list indicates the levels of aggregation possible for the selected basic variable. The bottom-most level on the list is the lowest level possible which is usually the level at which the data is stored in secondary memory (data records). The names in the map list are descriptive as to the level they represent (e.g. county).

This scheme of presentation of basic variables has several distinct advantages some of which will be indicated below. First of all, it presents a list of variables in an intelligible uncluttered fashion. In fact the list is larger and more intelligible than many

---

[14] There may not be any secondary attributes in which case the list is null and is not displayed.

[15] Whenever a list is displayed for light pen selection default values are preselected. Selection of an entity with the light pen causes it to be brightened and the previous selection to be unbrightened.

alternative methods of presentation. Secondly it provides a convenient facility for automatic aggregation of data elements as described in section 1.3.1. For example if data is stored at the census tract level and the user selects county level the census tract level data must be aggregated to county level.

Finally an address consisting of a file number and a record number is needed to access data for a basic variable. In BUDS the record number is calculated. This saves time and secondary storage when compared to table look up methods. The mechanism for basic variable data address calculation is discussed in detail in section 3.2.2.

Satisfied with his selections the user completes the APPEND function by selecting the RETURN button of the PFKB. However, if the user wishes to view another page of basic variables because he is unsatisfied with any of the choices presented to him or for any other reason he can do so by two mechanisms. In either mechanism he must display a year list page and repeat the process described in this section. He accomplishes this by selecting either the SELECT NEW YEAR function or the DISPLAY YEAR LIST function (function numbers 15 and 16 in Fig. 2a). The former displays the year list page that the user last used whereas the latter displays the requested year list page as detailed above.

After selecting the RETURN button the selected basic variables if any are appended in the proper location. It is noted here that the return function could have been selected without having requested the directory. In either event the RETURN function performs the same function. The RETURN function is automatically engaged if the

number of accepted basic variables reaches the appending limit in which case there will be ten basic variables in the created variable.

2.2.4. The Variable Header

Before proceeding to discuss other functions we pause to discuss the role of the variable header.

The variable header is the information preceding the list of basic variable in a created variable and is used to control the type of display (Fig. 4a). Touching NEW of the phrase NEW NORMALIZATION CONSTANT with the light pen will brighten the phrase and afford the user the option of changing NEW to SAME and vice versa (Fig. 4b). The option is exercised by selecting the ACCEPT function and nullified by selecting the REJECT function of the PFKB. Selection of either button unbrightens the phrase and selection of the former also changes NEW to SAME and vice versa.

Selecting SAME NORMALIZATION CONSTANT will force the normalization constant for this variable to have the same normalization constant as the previous variable on the display queue. Normalization refers to the procedure which must be undertaken in order to properly shade the sectors. The normalization constant is computed subsequent to variable computation and in the following way.[16] The sector on the screen for which the variable has the greatest value is found. The density of shading dots in this sector is fixed a priori to be 72 dots per square inch and the density of shading dots in the other sectors will be in the same proportion as their value is to the greatest value. If this a-priori density is too large the computed

[16] See Section A.2 of the appendix for details.

shading will overflow the allotted display memory space. If this happens the a-priori density is decreased dynamically until all of the shading can be accommodated in the allotted display memory space. Thus if sector A is the sector with maximum value and sector B has half the value as sector A then the density of shading dots in sector B will be half of that in sector A.

Normalization must be undertaken for each new normalization constant variable because of the following:

1. The value of a variable in each sector has a wide range of possible values and the density of dot shading is proportional to the value of the variable in each sector. Absolute values range from 10^{-128} to 10^{127}.

2. Since variable definition is under user control it is possible that negative variable expression values could be generated. This increases the range of possible values still further and introduces the problem of shading for negative values.

3. The amount of computer memory which can be dedicated to display or picture definition is limited.

4. The number of display items which can be placed on the screen is limited.

5. Physical damage to the screen is possible. If too many objects intersect at the same point on the screen, a hole will be burned in the phosphor of the screen.

Therefore, if normalization is not adapted to the variable values being displayed, then with a very high probability either there will be no shading at all or there will be too much shading.

Many situations arise in which the user will want to make

comparisons of two or more distinct pictures. To make appropriate comparisons he will often want to use the same normalization constant for the pictures he is comparing. Typically these comparisons are to be made over time or location or both. For example he may wish to compare the relative number and distribution of professional people in 1960 in San Jose with those in San Francisco. Or he may want to compare the relative number and distribution of professional people in San Jose for the years 1960 and 1970.

Automatic facilities help to insure that meaningless results do not occur. For example, the system will not allow the first variable on the display queue to be a "same normalization constant" variable since there is no variable preceding it on the display queue. If a "same normalization constant" variable should become the first variable on the display queue by one of several mechanisms it will automatically be changed to a new normalization constant variable. The system will not allow a variable to be a "same normalization constant" variable if the map of this variable is incompatible with the map of the preceding variable on the display queue. In fact if the map of any "same normalization constant" variable is incompatible with its predecessor it is automatically changed to a new normalization constant variable when transfer to VARDP is requested. Refer to section 2.2.12.

Touching NORMALIZATION CONSTANT with the light pen causes the message CHANGE TO LOW HIGH to appear on the screen. It also affords the user the option of changing from the density shading mode to the Low-High mode of shading (Fig. 4c) using the accept or reject buttons as described above. The Low-High mode of display uniformally shades all areas on the screen with characters in the following way.

All areas having the variable value less than the low real number are shaded with the low character. All areas having the variable value greater than the high real number shaded with the high character and all other areas are shaded with the medium character.[17] These characters and numbers may be changed at any time by the user. This is accomplished by touching the object to be changed with the light pen and entering a new value via the alphanumeric keyboard.

This mode of display allows the user to create a limited set of Boolean functions. Figure 5 is an example of a Low-High query. Touching LOW HIGH with the light pen affords the user the option of changing to the Dot Shading mode of display by using the ACCEPT or REJECT buttons to indicate his wish.

In reference to the automatic facility discussed above the system will not allow a "same normalization constant" variable to be preceded by a Low-High variable on the display queue. If the user attempts to change a "new normalization constant" variable preceded by a Low-High type variable to a "same normalization constant" type his attempt will be aborted by a bleep. If a variable is changed to a Low-High type and the next variable is a "same normalization constant" variable the latter will be automatically changed to "new normalization constant."

2.2.5. The Delete Variable Function

The user may delete any created variable whose definition appears on the screen by selecting the DELETE VARIABLE function (number 18). This causes the message DELETE VARIABLE to appear

---

[17] The medium character is the character between the real numbers in Fig. 4c. The low character and low real number are on the left of the medium character while the high real number and high character are on its right.

in the lower right corner of the screen. Touching any part of the variable to be deleted with the light pen causes the variable to be deleted from the display queue. The message and the deleted variable are then removed from the screen. Removal of a variable from the display queue causes the display queue to be compacted in the following manner. If the deleted variable is the last variable on the display queue, the number of variables on the display queue is simply decremented by one. Otherwise all of the variables following the deleted variable on the display queue are moved up and the number of variable is decremented by one. If the deleted variable is a "new normalization constant" type and the next variable on the display queue is a "same normalization constant" variable, "same" is automatically changed to "new."

If the message (the only non-variable object on the screen) is touched with the light pen, the function is aborted with a bleep. This also provides a way of retracting an erroneous function selection.

2.2.6. The Delete Basic Variable Function

The user may delete any basic variable appearing in the definition of a created variable which is currently displayed on the screen by selecting the DELETE BASIC VARIABLE FUNCTION (number 24 in Fig. 2a). This causes the message DELETE BASIC VARIABLE WITH LIGHT PEN to appear on the screen. The user then touches the basic variable to be deleted with the light pen and the selected basic variable is brightened. This selection is then accepted or rejected using the ACCEPT or REJECT buttons of the PFKB respectively. The former deletes the basic variable from the variable and removes the message.

The latter unbrightens the basic variable and the user must select another basic variable. However selection of an object other than a basic variable such as the message or part of a header causes the function to be aborted with a bleep removing the message in the process. This also provides an escape mechanism from a erroneous function selection.

2.2.7. The CHANGE VARIABLE POSITION on Display Queue Function

The user can reorder the variables on the display queue in any way he sees fit by selecting the CHANGE VARIABLE POSITION on the display queue function (number 20). This causes the message MOVE VARIABLE {0} to { } to appear in the lower right corner of the screen. The user then must input two numbers. The first number is the queue number of the variable he wishes to move while the second in the new position or new queue number he wishes that variable to have. These numbers replace the 0 and blank in the above message so the user can verify and change the numbers he is inputting. If either of these numbers is 0 or greater than the number of variables on the display queue the function is aborted with a bleep and the message is removed from the screen. Otherwise the message is removed and appropriate action is taken.

Automatic checking occurs to change a same normalization constant variable to a new normalization constant variable whenever it is necessitated by the movement of variables on the display queue.

This function is important for at least two reasons. It allows the user to order the variables chronologically or to some other specification. Secondly the user may wish to make comparisons using the same normalization constant and the computer cannot predict in

advance the values variables will take. Situations arise where a same normalization constant variable using the normalization constant as its predecessor will have too much shading and violate one of the constraints discussed in the section 2.2.4 necessitating reordering.

2.2.8. The DISPLAY VARIABLE # in SCREEN LOCATION # Function

If the user wishes to display the definition of a variable which is not currently on the screen to review its definition or modify it in some way he selects this function (number 25 in Fig. 2a) which causes the message DISPLAY VARIABLE {0} IN SCREEN POSITION { }. The user then inputs two numbers a queue number and a screen location which replace the zero and blank respectively. If the variable is already on the screen or if either number is inconsistent the function is aborted with a bleep. If a variable is currently on display in the desired screen location that variable is either deleted from the display queue or stored in secondary memory as described in section 2.2.4.

2.2.9. The COPY VARIABLE # to SCREEN LOCATION # Function

This function (number 25 in Fig. 2a) allows the user to make an exact duplicate of a variable residing on the display queue resulting in duplicate variables residing on the display queue. It allows the user to define a variable which differs in a small way from another variable without having to define that variable step by step.

Selection of this function causes the message COPY VARIABLE {0} TO SCREEN LOCATION { } to appear in the lower left corner of the screen unless ten variables already reside on the display queue.

In this case the function is aborted with bleep. The user then must input, using the alphanumeric keyboard, the queue number of the variable to be copied and the screen location in which he wishes the copies variable to appear. These numbers replace the zero and blank in the message respectively. The message is removed from the screen and the numbers are checked for incon- inconsistency. If either number is inconsistent the function is aborted with a bleep. If a variable is currently displayed at the desired screen location it is either deleted from the display queue or stored in secondary storage as discussed in section 2.2.4.

2.2.10. The Variable Expression and its Editing

The variable expression of a created variable is used to combine the basic variables in an arithmetic expression (Fig. 8a). This expression is then evaluated and the shading (Fig. 8b) is computed using the results of this expression. When basic variables are rearranged in a created variable either through appending or deletion letters are assigned to basic variables. The first variable below the variable header corresponds to the letter A and so on. This correspondence is used in the variable expression.

The grammar of the variable expression follows the convention used in Fortran.[18] It is however limited to 68 characters. Unlimited nesting of parentheses is permitted subject to the length constraint. A set of seven unary functions as well as the standard set of arithmetic operators (**,*,/,+,-) has been implemented. The functions

[18] See section A.3 for grammar.

and their expression codes are sine (SN), natural logarithm (LN), exponential (XP), square root (RT), hyperbolic tangent (HT), absolute value (AS) and cosine (CS).

The variable expression[19] of any created variable may be edited by touching any part of it with the light pen. The light pen is deactivated throughout expression editing. Touching an expression character causes a cursor (an arrow) to be placed under that character and a message to appear on the screen. The cursor always points to the position in the variable expression at which editing takes place.

There are two expression editing modes: the append mode and the overwrite mode. In the append mode the message EDIT EXPRESSION-APPEND appears in the lower left corner of the screen. If there are no characters in the variable expression (EXP = being the only entity below the basic variables) the append mode is automatically engaged and the cursor placed at character position 0. Any character (except space) may be entered into the variable expression via the alphanumeric keyboard. Each character so entered is inserted just after the character pointed to by the cursor (hence the append mode) and the cursor moved one position to the right. END is selected to exit from the append mode and the overwrite mode is automatically engaged. However, the append mode cannot be disengaged unless there is at least one character in the variable expression. In the append mode the cursor may point to character positions 0 through N, where

[19] The phrase EXP = appears as soon as basic variables are appended onto the header of a variable.

N is the last character in the variable expression. The append mode is automatically disengaged if the expression reaches its full complement of characters and the overwrite mode engaged.

In the overwrite mode the message EDIT EXPRESSION appears in the lower left corner of the screen. This mode is automatically entered by touching an expression character with the light pen when the variable expression needs editing. In the overwrite mode the cursor points to the character which may be replaced by another character through the alphanumeric keyboard. Hence the name overwrite Entering a character also moves the cursor one position to the right. In the overwrite mode the cursor can point to character positions 1 through N+1 where N is the number of characters in the variable expression. When the cursor points to position N+1, the overwrite mode will allow characters to be appended onto the end of the variable expression. The cursor is not permitted under any circumstances to point beyond character position 68. Selecting END in the overwrite mode terminates expression editing while selecting the APPEND function on the PFKB causes expression editing to enter the append mode.

Selecting the cancel key of the alphanumeric keyboard in either mode deletes the character pointed to by the cursor. The cursor is not moved unless it is pointing to the last character and in the append mode. In this case it is moved one position to the left.

The backspace key and space bar of the alphanumeric keyboard moves the cursor one position to the left and one position to the right respectively subject to the position constraints of the particular editing mode. Any attempt to violate these constraints is ignored.

2.2.11. The RESET DISPLAY QUEUE Function

This function (function 26 in Fig. 2a) allows the user to delete all variables from the display queue.

2.2.12. The DISPLAY VARIABLE Function

Selection of this function (number 12 in Fig. 2a) while in the variable definition and editing mode signals the end of variable definition and editing and indicates to the computer that the user wishes to calculate and display the variables he has created. The variables currently on display are stored on the disc and each variable on the display queue is checked in the following manner.

Any variable on the display queue without basic variables is automatically selected from the display queue and treated as though the DELETE VARIABLE function (section 2.2.5) was applied to it. Each variable is also subjected to a map consistency check and a legal expression check. If there are no errors transfer to the VARDP programs is automatically accomplished. Otherwise messages indicating which variable has which error is displayed on the screen. These error message are acknowledged by selecting the CONTINUE function on the PFKB and must be eliminated before the system will allow transfer to the VARDP programs to be accomplished.

Illegal expressions have syntactical errors while map inconsistency errors are one of two types. Either the map for a basic variable is inconsistent with the display map (section 1.3) or the display map is a pseudo map (discussed in the next two paragraphs).

The display map for a created variable is the lowest map level in a created variable. For example the created variable in Fig. 8a has two maps represented in its definition, MTC zones and City.

Since MTC zones can be aggregated to city level in San Francisco the MTC zones map is at a lower level than the City map (section 1.3). Therefore the display map will be MTC zone (Fig. 8b).

A pseudo map is a map which is defined relative to another map which has its two dimensional definition stored in secondary memory. This is not true for a pseudo map hence a picture such as the one in Fig. 8b cannot be made for a pseudo map. However pseudo maps are extremely useful because they allow the user to define his own map. For example, city in Fig. 8a is a pseudo map. In this instance it was necessary to use a pseudo map in order to form a per unit (percent) variable.

2.2.13. The STOP Function

This function (number 27 in Fig. 2a) indicates to the computer that the user wishes to terminate use of BUDS. The state of BUDS and the picture on the screen is stored in secondary memory so that the user may return at anytime and continue at the point of termination.

2.3. Map Manipulation and Variable Display Functions

The set of functions 0 through 14 constitute the map manipulation and variable display functions.[20] Figure 2b illustrates the accessibility order of these functions. These functions provide a flexible, easy to use yet sophisticated set of tools for map manipulation. We begin by giving some motivation for the strategies taken and data structures adopted.

Consider two map levels, a county level map and a census tract level map. At our disposal is a CRT whose effective area is 12" x 12".

[20] The Stop function is also part of this set. It has the same function as described in section 2.2.13 and is not discussed here.

Also, remember that one of the system objectives is to display the data in an intelligible fashion.

The screen is large enough to adequately display the nine county San Francisco Bay Area at the county level. However, this is not true of the census level map. There are 1058 census and pseudo census tracts[21] in this area for 1970 and the relative sizes in some cases are vastly different. At least two reasons exist for not displaying the entire map of this detail on the screen.

1. When one is interested in this level of detail, he is generally interested only in a small portion of the map. In fact, one would be hard pressed to simultaneously keep track of such a large number of areas.

2. Because of the finite screen size most of the locational data would be lost. Indeed, whole portions of the map would be reduced to a series of adjacent points indistinguishable from one another.

The best that one can do with a map larger than 12" x 12" is to use a "window" (i.e., display any 12" x 12" portion of the map.) That leaves the problem of selecting which portion of the map to window. In compliance with the system objectives of speed, intelligibility and ease of use the following scheme was adopted.

The entire area[22] is partitioned into 12 inch squares called subareas. The lines (boundaries) on the map within these subareas are stored in secondary storage and are referenced (pointed to) by the corresponding subarea. Thus by choosing a subarea the user can

---

[21] 1970 U.S. Census of Population and Housing.

[22] See section 3.3.2 for implementation details.

directly display the portion of the map associated with that subarea without having to sort through the entire map. This still leaves the problem of selecting the proper subarea and also of displaying any 12 inch square of the map, not just those corresponding to a subarea. Because subareas are 12 inches square it is clear that at normal magnification only four subareas need to be referenced in order to display any 12" square (window) of the map. At a magnification of .5 a maximum of 9 subareas must be referenced. Since the map data must be stored in secondary memory this data structure greatly reduces retrieval time as compared to retrieving the entire map. It also, reduces the amount of data which must be processed by the computer thus reducing computation time by the same factor.

To enable the user to easily choose the area of his interest some sort of global map of the metropolitan area in question must be presented to him. This map is called a Global Map. The Global Map consists of the outline of the metropolitan area together with locational features (transportation, rivers, etc.) and a set of points. Each point corresponds to the center of a subarea partition. The maximum size of the global map is 12" x 12" and therefore fits entirely on the CRT. The mechanism by which the user uses the global map to zoom in on the area of interest is described in the following sections.

It is convenient to divide the map manipulation and display phase into a global map phase and a map phase. The former is concerned with selecting a window while the latter deals with displaying the window and shading it.

Having transfered to the map manipulation and display phase from the variable definition and editing phase the computer first determines if any variables reside on the display queue. If there are no variables on the display queue a message to that effect is displayed on the screen. The user's only possible course of action is to acknowledge the message by selecting the DEFINE VARIABLES function (number 4 on the PFKB) and transfer back to the variable definition and editing phase. However if the display queue is not empty program control enters the DISPLAY VARIABLE # function[23] (number 14 in Fig. 2a).

For the sake of continuity it is assumed that the global map for the requested variable is on the screen leaving the discussion of this function for section 2.3.5.1. It is noted here that if the global map is trivial (i.e. the entire area is contained in one subarea) no global map is displayed. The global map phase in this case merely serves to reset certain parameters as described in the text. While the global map is on the CRT program control is said to be in this global map phase. Functions 0 through 4 of the PFKB are allowable in this phase. Some of the functions allowable in both the global map phase and map phase have different meanings. These will be indicated in the following sections.

2.3.1. Selecting a Window Using the Global Map

As indicated above the global map is used to make a window selection whereas fine tuning is available during the map phase.

Preliminary window selection is made by selecting a subarea. This is accomplished by touching one of the dots corresponding to

[23] See section 3.3.1 for details.

a subarea with the light pen. This causes the previously selected (or default subarea) to be unbrightened and the dot of the selected subarea to be brightened. The bright dot indicates the subarea containing the center of the window. This subarea is called the master subarea (MSA). Selection of any object which is not a subarea such as the transportation network, etc. is ignored. Further refinements can be made by using the MOVEX and MOVEY functions.

2.3.2. The MOVEX and MOVEY Functions in the Global Map Phase

Positioning the window so that it overlaps two or more subareas is accomplished through the use of the MOVEX and MOVEY functions (numbers 2 and 3 in Fig. 2a). Selecting the MOVEX function causes the message MOVEX {0} to appear on the lower left side of the screen. The user then inputs the number of raster units[24] by which he wishes the window to be moved via the alphanumeric keyboard which replaces the zero in the message. A positive entry moves the window to the right whereas a negative entry moves the window to the left. The user may input any integer from -32768 to 32767. Using this integer the computer then determines which subarea (or new master subarea) contains the center of the window (the window may straddle more than one subarea). If the desired move attempts to locate the center of the window in a nonexistent subarea (i.e. takes the center of the window off the map) the function is aborted with a bleep. If the move is legitimate the previous master subarea dot is unbrightened and the dot corresponding to the new master subarea is brightened. In addition

[24] The IBM 2250 uses raster units to position the beam. 1023 raster units equals 12 inches and a subarea is 1023 raster units square. This number which the user inputs does not correspond to the physical distance across the global map but to the physical distance on the map itself. Therefore on the global map 1023 raster units corresponds to the distance between two horizontally adjacent dots or the distance between representations of the centers of two adjacent subareas.

the coordinates in raster units of the center of the window relative to the center of the raster subarea are displayed in the lower right corner of the screen. The MOVEY function is similar to the MOVEX function with up corresponding to right and down corresponding to left. Therefore it shall not be discussed further.

2.3.3. The DISPLAY TRANSPORTATION Function

If transportation facilities are being displayed on the screen, selection of this function (number 2 in Fig. 2) will remove them from the screen and vice versa (Fig. 7c). Removing them reduces clutter while displaying them may aid in proper location of the window.

2.3.4. The DEFINE VARIABLES Function

Selection of this function (number 4 in Fig. 2a) indicates that the user wishes to return to the variable definition and editing phase. This transfer is made after storing the state of the map manipulation and variable display phase in secondary memory.

2.3.5. The Map Phase

During this phase the window is displayed on the screen. It is in this phase that the form of the map may be manipulated and variable expression and shading computed. Consequently, all results are derived in this phase. Functions 0 through 14 (in Fig. 2) are allowable in this phase. The display transportation and define variables function operate exactly as described in sections 2.3.3 and 2.3.4 during the global map phase. Therefore their description will not be repeated here.

2.3.5.1. The DISPLAY VARIABLE # Function

Having entered this function (number 14 in Fig. 2a) either through

user selection or transfer from the variable definition and editing functions the message NVDQ = 7, DISPLAY VARIABLE {0} is displayed on the screen. In this case the number of variables on the display queue (NVDQ) is 7. If NVDQ had some other value that number would appear instead of 7. The user then enters the queue number of the desired variable which replaces zero in the message. If this number is inconsistent (zero or greater than NVDQ) a bleep is sounded and the user must input another queue number (QN).

Once a proper queue number has been entered the computer determines whether or not that variable is currently on display. If it is the function is aborted.[25] The computer then determines the state of the variable with this queue number. If the shading has been computed the picture (map and shading) is retrieved from secondary memory and displayed. Program control then enters the map phase. If the shading has not been computed the computer ascertains whether the variable is a "same normalization constant" and also if a map for this variable has been constructed. If it is a "same normalization constant" variable and if the shading for the previous variable has not been constructed an error message to that effect is put on the screen. In this case the user must input another queue number repeating the above process.

If the shading of the previous variable (the variable with queue number QN-1) has been computed a message indicating that this is a "same normalization constant" variable and whether or not the map for

[25]This siutation cannot occur if transfer from the variable definition and editing had just been accomplished.

this variable had been constructed is displayed. The message also asks the user if he wishes to use the same window as the previous variable. He respones by selecting YES or NO with the light pen and selecting the CONTINUE/ACCEPT function. This option is useful if one wishes to compare different variables over the same sectors. At this point three actions are possible.

1. If the user wishes to use the window of the previous variable it is retrieved from secondary memory and displayed. Program control then enters the map phase.

2. If the user does not wish to use the window of the previous variable then either one of two cases are possible.

a. If the window for this variable was previously constructed, it is retrieved from secondary storage and displayed. Program control then enters the map phase.

or b. If the window for this variable is not constructed the program control enters the global map phase.

If the variable is not a "same normalization constant" variable and the window has been constructed it is retrieved from secondary memory and displayed. Program control then enters the map phase. Otherwise program control enters the global map phase.

2.3.5.2. The DISPLAY NEXT VARIABLE Function

The user selects this function (number 13 in Fig. 2a) if he wishes to display the next variable (i.e. the variable with queue number QN+1). Having selected this function the queue number of the variable currently on display is incremented by one and compared to NVDQ. If it is greater than NVDQ, the message YOU HAVE EXCEEDED nvdq,

THE NUMBER OF DEFINED VARIABLES is displayed on the screen. In the message the actual integer value of NVDQ is displayed in place of nvdq. After acknowledging the message with the continue button the message is removed from the screen and program control is returned to the point prior to entering this function with the original QN.

If the new queue number is proper the actions described in section 2.3.5.1 are taken.

2.3.5.3. The COMBINE AREAS Function

This function (number 11 in Fig. 2a) allows the user to combine individual sectors into larger units called blocks. Blocks themselves may be combined with other blocks or individual sectors forming still larger blocks. Common boundaries of sectors within a block are removed and from the user's point of view the computer treats a block as though it were an independent unit (i.e. an individual sector). Figure 11 shows the results of combining several sectors into two distinct blocks.

Having selected this function a list of areas to be combined is initialized. The following sequence of actions is possible:

I. The user then has one of three options: add an area[26] to the list of areas to be combined or either select the reject or return buttons of the PFKB.

A. Adding an area to the list of areas to be combined

All areas on the list which were computed have their identifiers[27] and shading brightened to distinguish them from areas not on the list.

---

[26] An area may be an individual sector or a block.

[27] A sector identifier is an asterisk within the boundaries of the sector. Each sector has at least one identifier.

To add an area to the list the user touches either an identifier or shading dot of the desired area with the light pen. If the area is already on the list or if the entity selected was a boundary the selection is ignored. Otherwise the area is added to the list and its identifier and shading is brightened. The user then must verify his selection using the ACCEPT or REJECT buttons.

The REJECT button removes the selected area from the list and unbrightens its identifiers and shading. Program control then goes back to I.

The ACCEPT button transfers program control back to I.

B. Selecting the REJECT button causes the selected areas to be unbrightened nnd the function to be aborted. Program control returns to the map phase.

C. Selecting the RETURN button

Selecting the RETURN button causes the areas on the list to be combined (unless the list is empty). That is the areas on the list are combined into a block and all common boundaries are removed. The window is frozen[28] and program control is returned to the map phase. It is noted here that the shading must be recomputed since the map has been altered.

2.3.5.4. The REMOVE AREAS Function

This function (number 6 in Fig. 2a) allows the user to remove areas from the window. Having entered this function a list of areas to be removed is initialized.

---

[28] When a window is frozen only the areas (partially or entirely) appearing on the screen at the time of freezing can be displayed until the window is unfrozen. The functions MOVEX, MOVEY as well as all other functions will operate, however no new sectors can be displayed.

I. The user then has the option of either selecting a area with the light pen or selecting the RETURN button of the PFKB.

A. Selecting an area with the light pen.

An area is selected by touching either one of its identifiers or its shading with the light pen. The selection of some other object is ignored. If the area is not already on the list it is added to the list and its identifiers and shading are brightened. If it is already on the list it is removed from the list and its identifiers and shading are unbrightened.

In either case the user must then select either the CONTINUE or the RETURN button of the PFKB. Selecting the CONTINUE button returns program control back to I. Selecting the RETURN button transfers control to B.

B. Selecting the RETURN button

If the remove areas list is empty the function is aborted. Otherwise all of the areas on the list are removed from the window and the window is frozen. It is noted here that the shading must be recomputed since the map is modified. In either event program control is transfered to the map phase.

This function has two important uses. First by removing areas from the screen it removes clutter thus enabling the user to concentrate on fewer areas. In the process it also speeds up computation of shading since less shading must be computed. Secondly it allows the user to remove the most densely shaded areas so that secondary effects can be observed which are not visable because of normalization (Fig. 10 and see section 2.2.4). Recall that shading is computed on a relative basis. If an area on the screen has a value which is

orders of magnitude greater than that of other areas, these other areas will have no shading.

2.3.5.5. The MOVEX and MOVEY Functions

These functions (number 2 and 3 in Fig. 2a respectively) allow the user to move the window horizontally and vertically over the map enabling him to fine tune his area of interest. Having selected the MOVEX function the message MOVEX {0} appears in the lower left corner of the CRT. The user then inputs an integer which replaces the zero in the message. The integer represents the number of raster units the user wishes to move the window. A positive integer moves the window to the right whereas a negative integer moves the window to the left. A distance of twelve inches corresponds to 1023 raster units. The user may input any integer between the values of -32768 and 32767. If the user inputs an integer which attempts to move the center of the window into a nonexistent subarea (off of the map) the function is aborted with a bleep.

The MOVEY function is similar to the MOVEX function with up corresponding to right and therefore is not discussed here.

If the window is frozen, sectors which were not on the CRT at the time the window was frozen will not be displayed.

2.3.5.6. The MAGNIFY Function

This function (number 5 in Fig. 2a) allows the user to magnify the window using a magnification factor between .500 and 9.999. Having entered this function the message MAGNIFICATION FACTOR 1.000 with the previously selected magnification factor (1.000 in this instance) appearing in the lower left corner of the CRT and a tracking

box appears at the previously selected point of magnification. The magnification factor is changed by using the alphanumeric keyboard. However the user must select the point about which he wishes magnification to take place before terminating the magnification factor definition. He does this by moving the tracking box with the light pen. If the magnification factor is out of the range defined above the function is aborted with a bleep. If the point of magnification and the magnification factor have not been changed the function is aborted without a bleep. Otherwise the map is magnified with the new magnification factor about the new point of magnification. The magnification factor is relative to the standard size map and not to what is on the screen.

It is noted here that this function does not alter a frozen picture except by magnification. That is sectors which were not in the frozen picture will not be displayed. Also note that a magnification factor of 1.000 displays the window as it is in secondary memory (no magnification).

This function is useful for several reasons. Magnification can improve perception or clarity. By using a magnification factor of .500 it can be used to bring more sectors on the screen. This is especially useful for rural areas where census tracts tend to be rather large. The increased area of coverage may also aid in proper window positioning by giving the user a larger detailed view of the map.

Note that magnification factors outside the range (.500, 9.999) cannot be obtained by repeated use of this function.

2.3.5.7. The RECONSTRUCT MAP Function

This function (number 8 in Fig. 2a) unfreezes the windows. That is all removed sectors are displayed and all combined sectors are uncombined. If the picture was already unfrozen this function is ignored.

2.3.5.8. The GLOBAL MAP Function

This function (number 7 in Fig. 2a) transfers program control to the global map phase. Selection of this function also unfreezes the window and resets the magnification factor to 1.000 and the magnification point to the center of the window.

This function is useful in that it allows the user to reposition the window in a gross sense. For example, the user while making queries over the nine county bay area may wish to move the window from the San Francisco area to the San Jose area. It may also help him to determine where the window is currently located relative to the global map.

2.3.5.9. The DISPLAY VARIABLE Function

After being satisfied with the window this function (number 12 of the PFKB) is selected to compute and display the shading. If the variable expression is not computed it is calculated otherwise the appropriate data is retrieved from these computation and shading computation and display are commenced.

If the shading has already been computed and is currently on display selection of this function will remove only the shading from the screen. A subsequent selection of this function will restore the shading. This latter feature aids in comprehension especially

when used in conjunction with the DISPLAY MAP and DISPLAY TRANSPORTATION functions.

2.3.5.10. The DISPLAY MAP Function

In the map phase this function (numbers 10 in Fig. 2a) simply removes the boundaries of the areas in the window from the CRT when it is on display. Subsequent selection restores the map. This function is useful for comprehension especially when used in conjunction with the DISPLAY VARIABLE and the DISPLAY TRANSPORTATION functions.

2.3.5.11. The IDENTIFY Function

This function (number 9 in Fig. 2a) enables the user to identify by name each area on the screen using the light pen to indicate the area of interest. Having selected this function the user has the option of selecting the RETURN button on the PFKB or selecting another area to be identified with the light pen.

Selecting the RETURN button transfers control out of this function whereas selecting an area on the screen causes its identifiers and shading (if computed) to be brightened and the full name of the area is typed on the typerwriter. The user then must select either the RETURN or the CONTINUE button on the PFKB. Selection of either button unbrightens the selected area. The CONTINUE button allows the user to repeat the above process whereas the RETURN button transfers control out of this function.

2.3.5.12. The PRINT Function

Selection of this function (number 10 on the PFKB) will be

acknowledged only if the shading has been computed. Hardcopy quantitative results are obtainable in the following way. Having selected this function a message appears asking if the variable definition is to be typed on the typewriter. The user then responds by selecting either the ACCEPT or REJECT buttons on the PFKB. Selecting ACCEPT button causes the variable definition to be printed on the typewriter exactly as it appeared on the CRT when it was defined.

Having completed printing or if the REJECT button was selected a message is printed on the typewriter asking if variable values for individual or all areas on the CRT are desired. The user must select either the ACCEPT or REJECT buttons on the PFKB. Selecting ACCEPT causes the computer to go into the individual area mode. In this mode the actions of computer and user are identical to those of the IDENTIFY function (section 2.3.5.11) with the exception that the value of the variable for each selected area is printed along with its name. This procedure is repeated until the RETURN button is selected.

If the REJECT button is selected then the full name and variable value of every area on the CRT is typed on the typewriter. Program control then leaves this function.

CHAPTER 3

DATA STRUCTURES AND IMPLEMENTATION

3.1. Introduction

There are five types of files in BUDS: a master file, basic variable data files, map files, transportation files and a picture data file. There is only one master file and only one picture data file whereas there may be several files for each of the other files types. The master file contains the information necessary to integrate and access all of the files. The picture data file contains the state of BUDS and all of the data associated with each created variable. The names of the other files describe their content. This chapter described the files in terms of their function, content and implementation with the following exceptions

The structure of basic variable data files was discussed in section 1.2.6 and throughout the text. Therefore further discussion would not be worthwhile. The picture data file is discussed in section A.5.

In what follows -10000 serves as an end of logical record and -10000-NREC serves to indicate that the logical record continues at record NREC. This convention made it possible for variable length logical records.[1] The symbol EV is the acronym for entity value. Every entity put onto the CRT has an entity value, an integer number, associated with it. The entity selected with the light pen is communicated to the program via its entity value. X and Y are the

---

[1]This is not really true since the records are already logical records and hence these record markers may be considered as another level of logical records.

acronyms for the x and y coordinates of the object to be constructed on the CRT. DP and LP are the acronyms for disk record pointer and line pointer. Disk record pointers are pointers to records within the file while line pointers point to a particular cell in the record. This cell is the first cell of information pointed to by the pair DP and LP. If DP appears by itself LP is implicitly assumed to be one, the first cell in the record.

3.2. The Master File

The master file contains all the information necessary to access data from the computer data bank. In particular it contains the necessary information to create variables and compute the variable expression.

3.2.1. The Directory Paging Structure

Section 2.2.3 discussed the selection of basic variables from the directory. This section discusses the structure enabling the selection of basic variables in that manner. Consider Fig. 12. As mentioned in section 2.2.3 selecting a year list page produces a list of years. Selecting a year from this year produces a list of ten pointers. Each pointer in this list points to a record in the master file. Each of these pointers has a role in selecting a basic variable from the directory. The names of these pointers are

UVNDP - urban variable name disk pointer
UVPDP - urban variable primary attribute disk pointer
UVSDP - urban variable secondary attribute disk pointer
UVMDP - urban variable map list disk pointer
VPNTP - variable pointer array disk pointer
VCDNP - variable control array disk pointer
SDCOD - secondary attribute decoder disk pointer
LDCOD - map level decoder disk pointer
PXUND - primary attribute entity value to urban name disk pointer
PXPAD - primary attribute entity value to primary attribute disk pointer

and the function[2] of each is described in the following sections. Figure 13 is a detailed version of Fig. 12. The first record of MASFL contains as its first word, N, the number of year list pages followed by N disk pointers. Each of these pointers points to a record used to construct a year list page. If there is not enough room in the first record for N integers, the list would be continued in disk record NREC.

Currently there are only two years 1960 and 1970 are the only year list page in BUDS.

3.2.1.1. The Basic Variable Page Construction

The words in a page of basic variables such as those in Fig. 6 are constructed using the disk pointer UVNDP, UVPDP, UVSDP and UVMDP. In section 2.2.3 it was pointed out that an urban variable consists of a name, a list of primary attributes, a list of secondary attributes and a list of maps. A basic variable consists of a name and one element from each of the three lists. In the example shown in Fig. 6, population age is the urban variable name. 0-19, 20-24, 25-34, 35-44, 45-65 and 65+ are its primary attributes. Total, nonwhite, spanish surname and white are its secondary attributes and county, city and MTC zone comprise the list of maps over which this urban variable can be computed. The structures given in Fig. 14 are used to construct urban variable definitions on the CRT. UVNDP is the acronym for urban variable name disk pointer. UVPDP is the acronym for urban variable primary attribute disk pointer. UVSDP is the acronym for urban variable secondary attribute disk pointer. UVMDP is the acronym for urban variable map disk pointer.

[2] PXUND and PXPAD are described in section A.4.

The structures differ only in one detail. There are no entity values for urban variable names. Implementation demands that every object put on the CRT must belong to an entity and each entity must have an entity value. Variable names all belong to the same entity which has zero as an entity value. However each primary and secondary attribute and each map are separate entities with distinct entity values. There are at least two reasons for doing this. First the physical layout in Fig. 6 indicates that the selection of a primary attribute specifies the urban variable names. Hence the selection of an urban variable name is redundant. Secondly section 2.2.3 indicated that a basic variable consists of an urban variable name, a primary attribute, a secondary attribute and a map. The primary and secondary attributes are used to calculate the data address of the basic variable whereas the map indicates whether or not and how the data must be transformed (section 1.3.1). The next sections discuss how this is done.

This particular structure was selected because of the following reasons. First the character definitions of the words can be shared by any portion of the system that needs to access character definitions minimizing redunancy. However there is a much more important reason. This structure facilitates the addition or deletion of data. To delete data pointers to the character data to be deleted are found and eliminated by compacting that array (disk record). Since the disc records which are used as pointers are either 80 or 160 words long this task is trivial. This action would leave unused cells at the end of the record which could be used for the addition of data later. Also it is not necessary to remove character definitions,

but only their pointers. Addition of data may also be accomplished by appending character definition pointers onto the end of the last disk record. Of course this process for deletion or addition only accomplishes deletion or addition of basic variable definitions. Other pointers and data must be entered for addition of data (refer to the next sections). But nothing else is necessary for the deletion of data since if basic variables cannot be displayed on the CRT for light pen selection they are unavailable for selection.

3.2.2. Calculating the Basic Variable Data Disk Record Address

The structure in Fig. 15 is used to compute the selected basic variable disk record address and to display the proper secondary attribute and map lists for selection. This is accomplished in the following manner. The object selected by the light pen is identified via its entity value. When a primary attribute is selected its bias[3] is removed from its entity value yielding SPATT the acronym for selected primary attribute. SPATT is used as a pointer into VPNT the acronym for variable pointer array which in turn contains pointers into VCON the acronym for variable control array. The third column of VCON indicates which secondary attributes are associated with this urban variable. The fourth column of VCON indicates which maps are associated with this urban variable and if the urban variable data is stored in floating point or integer format. The next section indicates how these are used to produce lists which are displayed on

[3]Entity values are integers. Therefore to distinguish primary attributes from secondary attributes from maps in the map list each category is given a bias. Subtracting the bias from an entity value gives a parameter a unique identifier within that category.

the CRT.

The method of basic variable disk record address calculation is given via an example. Suppose that an urban variable has three primary attributes and five secondary attributes. Suppose that the data for all of the primary attributes are stored sequentially and in order for the first secondary attribute, then for the second, third, fourth and fifth secondary attributes. That is the data records are stored by varying the index over the primary attributes more frequently than the index over secondary attributes in a two index scheme.

Let DREC be the record in the basic variable data file containing the data corresponding to the first primary attribute and the first secondary attribute (i.e. the double index 1,1). Let the entity value with bias removed of this primary attribute be SPATT1. Then there will be 3 sequential entries in VPNT corresponding to each primary attribute all pointing to the same line in VCON which implies the entity values of primary attributes associated with the same urban variable must be sequential and in the order corresponding to the order of storage.

The disk record address of the selected basic variable is given as

RECORD = SPATT + Base Address + (SSATT-1) * OFFSET

Refer to Fig. 16 for pictorial details.

SSATT is the acronym for selected secondary attribute and has the value corresponding to the position of the selected secondary attribute in the list of secondary attributes for this urban variable. The order of secondary attributes in the secondary attribute list

corresponds to the sequence of storage in the basic variable data file. Refer to section 3.2.3.

This structure is both simple and efficient. Because urban variables usually have many modifiers this scheme usually requires less memory storage for list control and address calculation than more direct methods of addressing. Secondly Fig. 6 indicates that the directory is highly intelligible considering the quantity of information it represents. Current implementation restricts the total number of primary attributes on a page of basic variables at 80 and the number of urban variable names to 40. Considering screen size experience with the current system indicates that these numbers are adequate. Simple programming modifications can change these numbers but since the number of year list pages is limited only by the amount of secondary memory, it seems unlikely that a change will be necessary. Deletion of data is accomplished by placing minus one[4] in the appropriate locations in VPNT. Unless other primary attributes point to the same location in VCON these should also be set to minus one. Although these actions are not really necessary (refer to section 3.2.1.1), they are recommended for the convenience of data base management.

The addition of data involves the opposite of deletion and therefore is not discussed here.

3.2.3. Secondary Attribute Lists and Decoding Selected Secondary Attributes

SLIST is the acronym for secondary attribute list number. Each urban variable has a list of secondary attributes (sections 2.2.3

[4]Although any negative number will do minus one is currently being used and for the sake of clarity in data base manangement this convention should be continued.

and 3.2.1.1). This list number is obtained from column 3 of VCON as described in section 3.2.2. To retrieve the actual members of this list SLIST is used as a line pointer into a two dimensional array pointed to by the secondary attribute decoder disk pointer, SDCOD, as shown in Fig. 17. This two dimensional array then provides a disc record and line pointer to a one dimensional array. The word pointed to by the line pointer is the number of attributes in the secondary attribute list followed by indexes to these attributes. These indexes are the entity values of the secondary attributes minus their bias and are used to determine which attributes belong to the list corresponding to SPATT or SLIST. When a secondary attribute is selected using the light pen its index's position in this list is SSATT which was discussed in section 3.2.2.

3.2.4. The Map Lists and Map Level Decoding

As indicated in Fig. 15 the fourth column of VCON contains the map level list, MLIST, corresponding to the urban variable from which SPATT was derived. A bias of 20 is attached if the data for this urban variable is in floating point format. In what follows it is assumed that MLIST is unbaised. Figure 18 indicates the structure for determining the members of this list and decoding the light pen selected map level into a map file number, MFN.

MLIST is used as a line pointer into a two dimensional array pointed to by LDCOD. This in turn provides a disk and line pointer into another two dimensional array. Referring to Fig. 18 these two pointers yield in the first column the number of maps in this list followed by indexes. These indexes are the unbiased entity values

of the map descriptors presented for light pen selection (refer to section 2.2.3). The maps in this list indicate the map levels to which the basic variable data can be transformed (refer to section 1.3.1).

The second column contains the data file number, DFN, containing the selected basic variable data and the map file numbers, MFN, corresponding to the map list. Accepting a basic variable causes DFN and the selected MFN to be recorded and stored along with the basic variable disk record address (section 3.2.2) and other information necessary to construct the basic variable word definition. DFN and MFN are used for data consistency checks and data transformation (refer to sections 3.2.5 and 3.2.5.1). This structure allows a great deal of flexibility. First of all more than one data file can be associated with a page of basic variables. In fact more than one data file can be associated with an urban variable. That is the data for primary attributes of an urban variable can be stored in different basic variable data files. Therefore if a basic variable data file is full and it is desired to add data to the system the data may be entered into another basic variable data file.

Different lists of maps may be associated with a page of basic variables. As in the preceding remarks different map lists may be associated with the primary attributes of an urban variable. Thus the raw data associated with an urban variable may be stored in the computer with respect to different maps. For example data for one primary attribute may be stored at the census tract level while the data for another primary attribute in the same urban variable may be stored at the county level.

3.2.5. The Map - Basic Variable Hierarchy

The map - basic variable hierarchy as its name implies is the structure which links together map data (locational data) with basic variable data (urban data). This hierarchy can be represented as a directed graph as shown in Fig. 19.

Each node of the graph represents a map file number (or map) and at least one basic variable data file (BVDF). There is a one-to-one correspondence between each data element of a data record in the BVDF and each sector of the associated map. This correspondence is discussed in Chapter I especially section 1.2.6. The map file or for that matter the BVDF may not exist. In which case they are referred to as a pseudo map file or psuedo BVDF. Pseudo map files are maps whose correspondence to a BVDF exists in BUDS but whose two dimensional coordinates do not exist in secondary memory. Hence a pseudo map is unavailable for display as in Fig. 7b. However, using this correspondence transformation of data records from one BVDF to another is possible. Refer to section 1.3.1.

Because it is a simple matter to create a pseudo map the user can define his own map relative to another map. Future software expansion should exploit this possibility even further. For example, the process of defining a pseudo map may be considered as just another method of executing the COMBINE AREAS function (section 2.3.5.3) in which case the pseudo map would be available for display purposes.

Psuedo BVDF's do not exist in secondary memory and play no role in the current implementation except possibly for completeness. However they cost nothing to implement and will probably have a role in future software expansion. Internally pseudo files are distinguished from existing files in that the former have negative file numbers while the

latter have positive file numbers.

Two nodes of the map - basic variable hierarchy graph are connected if the map of one node is a refinement or cover of the map of the other as described in section 1.3. The direction of the arc indicates the direction of the transformation. The arc leaves the node corresponding to the map which is the domain of the transformation and enters the node corresponding to the resultant map or range of the transformation. Since these transformations are linear, transitivity implies that if there is a directed path between any two nodes there must also be an arc between these two nodes. Also as indicated in Fig. 19 the graph need not be connected. At present the depth of the graph (the longest directed path) is contrained to be 26 arcs. Simple software modifications can remove this restriction. However, as a practical matter it is difficult to see the need for depth beyond the current implementation which is discussed in section 3.2.5.1.

This section is concluded with an example using Fig. 19. The 1960 and 1970 Census tract level maps of the 9 County Bay Area are incompatible. Therefore there is no arc connecting these two nodes. However, the sectors of both maps can be aggregated using the transformation discussed in section 1.3 to form an MTC zone map. The MTC zone map can be transformed in a like manner into either a city map or county level map. Therefore by transitivity there is an arc from each of the census tract maps to the city and county maps. The city level map cannot be transformed into the county level map because some sectors of the city level map straddle county boundaries. Hence ther is no arc connecting these nodes.

The next section discusses the implementation which is somewhat more general than the one described here.

3.2.5.1. Implementation of the Map - Basic Variable Hierarchy

Figure 20 illustrates the details of the map basic variable hierarchy implementation. The map list disk record pointer, MLPT, and the data file list disk record pointer, DLPT, are kept in first record of the master file as shown in Fig. 13. As previously mentioned all maps and BVDF are referenced by their file numbers. This establishes a correspondence between the nodes of Fig. 19 and the cells labeled MFN and BVDFN in Fig. 20. That is MFN, map file number, corresponds to a map and BVDFN, basic variable data file number, corresponds to a basic variable data file.

NM_k in Fig. 20 is the acronym for the number of maps in BUDS for which a transformation as described in section 1.3.1 is applicable on the map MFN_k.[5] Equivalently NM_k is the number of arcs leaving the node corresponding to MFN_k in Fig. 19 plus one. MLPT points to the map list array. The map list array contains all map file numbers corresponding to all maps (including pseudo maps) stored in BUDS. Associated with each MFN in the map list array is a pair of pointers (DP and LP) and NM. These pointers point to a list of NM maps in the map relation array. The list of NM maps in the map relation array is headed by the map file number associated with the identity transformation followed by map file numbers corresponding

[5] This includes the identity transformation i.e. when no transformation is necessary. The minimum value of NM_k is one.

to the maps obtained via transformation. In graph theory if the subtree generated by the maps in this list is considered, the first map file number in this list corresponds to the parent node while the remaining map file numbers in the list correspond to son nodes or filial set of the aforementioned parent nodes (Fig. 19). Hence the name map relation array.

Associated with the parent node map file number in the map relation arrays is NS, the number of sectors in this map, and DFRL, the number of words in a logical record corresponding to basic variable data file associated with the parent node. Their use is discussed below. Associated with each son node map file number is a pair of pointers, DP and LP. These pointers point to the transformation definition which is discussed below.

This data structure is well suited for a map consistency check and for the determination of the display map. These concepts are introduced via an example. Consider Fig. 8. Figure 8a shows two basic variables each with a different map. Two questions arise: "Will the mixing of the data for these two basic variables make sense?" or equivalently "Is there a transformation relating the two maps (refer to section 1.3)?" And "If the maps are consistent which map should be used as the display map (Fig. 8b)?" From the preceding comments it is easy to see that the display map must not be a psuedo map and it must correspond to a parent node. For the maps to be consistent all other maps in the created variable must correspond to son nodes of the display map or to the parent node itself.

Attention is now focused on the transformation implementation.

In section 1.2.1 the polygonal subsets of a map were defined as sectors. In the implementation each sector of a map is assigned an integer called a sector number, SN. Sector numbers are used as indexes for a variety of activities. In section 1.2.5 a one to one correspondence between sectors and components of data records was established. Sector numbers serve as indexes into data records thus defining the correspondence. It is also natural to assign sector numbers in an orderly sequential fashion. For example consider the 290 sectors comprising the MTC ZONE map for the 9 county bay area Whenever possible sector numbers were assigned so that the sectors comprising an area were sequential. For example, sectors 1 through 40 comprise both the city and county of San Francisco while sectors 222 through 290 comprise Alameda county.

As mentioned above each map file number corresponds to a son node has a pair of pointers (DP and LP) which point to its definition relative to the map corresponding to its parent node. In the map relation array of Fig. 20 MFN_{10} is defined relative to MFN_2 via the transformation pointed to by DP and LP. These two pointers point to a table which has four entries for each sector[6] in MFN_{10}. Referring to Fig. 20 SN is a sector number for a sector in MFN_{10}. DP and LP are pointers into a table of pairs of sector numbers. NP is the number of pairs of sector numbers in the map of MFN_2 necessary to form the sector SN in the map of MFN_{10}. Using

[6] In the implementation if sector A of MFN_{10} is the same as sector A of MFN_2 then entries for sector A are unnecessary.

the notation of section 1.2.1 this is expressed as

$$S_{SN} = \bigcup_{i=1}^{NP} \bigcup_{j=SN_{1i}}^{SN_{2i}} S_j$$

where

$S_{SN} \in$ the map of MFN_{10}

$S_j \in$ the map of MFN_2.

Attention is now focused on transformation of basic variable data. In section 3.2.4 it was pointed out that when a basic variable is selected its disk record address, data file number (BVDFN) and map (MFN) are recorded. If the BVDFN and MFN of a basic variable are associated with different nodes of Fig. 19 then transformation of the data is necessary. Determination of whether or not transformation is necessary and how it is performed is carried out in the following way. The basic variable data file list array of Fig. 20 contains the data file numbers of all basic variable data files stored in BUDS. This table is searched for the BVDFN of the basic variable. Then the list of maps in the map relation array pointed to by DP and LP associated with this BVDFN is searched for the MFN of the basic variable. If it is the first in the list no transformation is necessary otherwise transformation is necessary. Continuing the above example the preceding equation indicates this transformation can be expressed as

$$DE_{SN} = \sum_{i=1}^{NP} \sum_{j=SN_{1i}}^{SN_{2i}} DE_j$$

where DE_j is the jth component of the data record in the BVDFN for this basic variable over the map corresponding to MFN_2

and DE_{SN} is the SNth component of the data record for this basic variable over the map corresponding to MFN_{10}

This section is concluded with remarks concerning the cells labeled NS_2 and DFRL of Fig. 20. NS_2 is the number of sectors in the map MFN_2 or equivalently the number of data elements in each data record of $BVDFN_2$. DFRL is the logical data file record length of the file $BVDFN_2$. The operating system of the computer has physical records of 320 words. Furthermore logical records cannot cross physical record boundaries. Therefore the proper choice of DFRL can avoid the waste of space on the disk sometimes at the expense of longer retrieval time.

One final comment, since pseudo files are distinguished from existent files by negative numbers -10000 could not be used as an end of record indicator. Instead 0 was used as an end of record marker in the map list array and the basic variable data file list array (refer to Fig. 20). NREC serves its usual function.

3.2.6. The Transportation Map - Map Hierarchy

Transportation maps are not as static as maps (with sectors). For example the transportation map for the nine county bay area in 1960 is radically different from that of 1970 yet the county map was unchanged over this period. Therefore a mechanism is necessary to link the proper transportation map to the proper map with respect to both time and location. The data structure to accomplish this is shown in Fig. 21.

TLPT the transportation list disc record pointer is kept in the first record of the master file as shown in Fig. 13. Transportation maps are referenced by their file numbers. The map file number of every map stored in BUDS which has one or more transportation maps (also stored in BUDS) associated with it is contained in the map with transportation list pointed to by TLPT. If the map with transportation list does not contain the display map file number no transportation map exists for that map. If the display map file number is contained in this list the two pointers (DP and LP) are used to access a table of transportation file numbers called the year transportation map list.

The first cell of this list contains NTF, the number of transportation file numbers in this list. An attempt to match the year contained in the basic variable which determined the display map is made in the year transportation map list. If unsuccessful there is no transportation map for the display map. If successful the transportation file number, TFILN associated with that year corresponds to the proper transportation map.

3.3. Map Files

As indicated in section 3.2.5 each map is contained in a separate map file with the exception of pseudo maps. Pseudo maps exist only through definition relative to another map and this definition resides in the master file. The first record of a map file contains six disk record pointers. They are

SADPT	Subarea map disc record pointer
SCDPT	Subarea sector center disc record pointer
SHDPT	Subarea sector shading disc record pointer
GMDPT	Global map disc record pointer
ALDPT	Subarea alignment disc record pointer
SNNPT	Sector name disc record pointer

Section 2.3. contains a general discussion concerning the use of maps. Therefore the following sections will be primarily concerned with implementation

3.3.1. The Global Map

As indicated in section 2.3 the CRT is only twelve inches square. Therefore it is impossible to display large maps with an acceptable degree of intelligibility. For example the 1970 census tract map of the 9 county bay area has 1058 census tracts and the area of the largest census tract relative to the smallest is well over 10^3. The best that can be done is to display a twelve inch window of the map.

The problem now is provide a mechanism enabling selection of this window which is simple to use and highly intelligible.

The global map provides this mechanism.

Because the Meta IV is a small machine with little primary memory and limited hardware capabilities the partitioning scheme described below was adopted in order to greatly reduce both the amount of data which must be retrieved from disc and the amount of computation necessary to display the choosen window. With this scheme the entire map need not be searched in order to display the window. The partitioning scheme is as follows: the entire map is partitioned into 12 inch squares like a checkerboard. Line segments (sides of map sectors) which cross partition boundaries are decomposed into two line segments with the separation point at the boundary. These partitions are called subareas. Subareas are stored in secondary memory and accessed via their subarea numbers. Refer to section 2.3 for other details.

The global map consists of the outline of the metropolitan area together with locational features (rivers, large bodies of water, etc.) and a set of points. Each point corresponds to a subarea. Using the global map the user is able to select any twelve inch window by selecting a subarea with the light pen and using the MOVEX and MOVEY functions as detailed in sections 2.3.1 and 2.3.2.

Figure 22 indicates the data structure with which the global map is stored in secondary memory. Logical data records are 320 words long. The first record is pointed to by GMDPT, the global map disk record pointer. The first word of this record is NL the number of lines, end of information or disk record chain address. If NL is the number of lines it is followed by a list of NL x coordinates, a list of NL y corrdinates and a list of NL modes. Mode i indicates the type of CRT beam instruction associated with the coordinates (x_i,y_i). For example if mode i = 3 the beam with the electron gun turned on will be displaced (x_i,y_i) raster units from the current beam position. Thus a line will be drawn from the current beam position a distance of $\sqrt{x_i^2+y_i^2}$ raster units. The end of outline definition is signaled by NL =-10000. The outline definition is followed by the subarea markers. The first word commencing the subarea markers is SAN which is a subarea number, end of information or disk record chain address. If SAN is a subarea number it is followed by absolute x and y coordinates indicating where the point should be placed.

3.3.2. The Subarea Data Structure

This section deals with the data structure in map files necessary to manipulate subareas. There are four interrelated structures:

(1) The subarea sector boundary data structure as its name implies contains the data necessary to construct sector boundaries (2) The subarea sector center data structure contains the data necessary to put an asterisk inside the boundaries of all sectors in the subarea. (3) The subarea sector shading data structure as its name implies contains the data necessary to calculate shading dot or character location. (4) The subarea alignment data structure contains the data indicating the subarea connectivity.

Each of these structures and their interrelationships is discussed in the next four sections.

3.3.2.1. The Subarea Sector Boundary Data Structure

Figure 23 illustrates the subarea sector boundary data structure. SADPT, the subarea map disc record pointer, points to the subarea disk record pointer array. The number of subareas, NSA, currently allowed in any map is 79. This corresponds to a 79 square foot map which seems adequate. However minor softwave and file modifications will remove this restriction.

The subarea numbers of the desired subarea is used as an index into the subarea disc record pointer array which contains the disc record pointer and, NLIN, the number of lines in the subarea or equivalently the number of entries in sector boundary array. In the sector boundary array x and y are absolute coordinates meaning move the CRT beam to position (x,y). Hence the range of x and y are integer values between 0 and 1023. Associated with each coordinate pair are two number $SN_{i,1}$ and $SN_{i,2}$. $SN_{i,1} = 0$ signals the software to move the CRT beam with the electron gas turned off to coordinates

(x_i,y_i). That is draw a blank line from the current beam position to (x_i,y_i). $SN_{i,1} > 0$ indicates that the line from (x_{i-1},y_{i-1}) to (x_i,y_i) is a boundary between sector $SN_{i,1}$ and sector $SN_{i,2}$. $SN_{i,2} = -1$ indicates that the aforementioned line is an exterior map boundary. That is it is a boundary only for sector $SN_{i,1}$.

The sector boundary array data structures has two main advantages. It requires less secondary storage and consequently less retrieval time and more flicker free objects can be put onto the CRT. Normally it requires six words to specify the endpoints of a line and the sectors for which it is a boundary. This data structure contains the same information with only four words (plus a little). This is accomplished in the following way. Before the subarea boundary data is entered into secondary storage it is processed by a simple program. This program starts at a line endpoint and tries to draw the longest line, along sector boundaries possible without picking up the pencil. This data is entered into the sector boundary array and the process repeated until all of the subarea data is accounted for. Although crude this procedure saves considerable amounts of storage space, retrieval time and display items. The saving of display items is especially important since it takes as much time to draw a blank line as it does a line.

The sector boundary array data structure also provides a convenient mechanism for the suppression of selected sector boundaries.

3.3.2.2. The Subarea Sector Center Data Structure

The subarea sector center data structure is illustrated in Fig. 24. SCDPT the subarea sector center disc record pointer points

to the subarea center disc record pointer array. The subarea number of the desired subarea is used as an index into this array to access a disc record and a line pointer pointing to the sector center data. Each entry consists of a triplet SN the sector number and CRT coordinates (x,y) for this sector. The coordinates are used to place an asterisk which is placed within the boundaries of the sector and used as a unique sector identifier. Sector boundaries cannot provide a convenient way to select a sector since in most cases at least two boundaries must be selected to specify a sector. This would be cumbersome for the user. Secondly implementation would be much more difficult and greatly degrade system performance.

3.3.2.3. The Subarea Sector Shading Data Structure

Figure 25 illustrates the subarea sector shading data structure. SHDPT, the subarea sector shading disc record pointer points to the subarea pointer array. The sector number of the desired subarea is used as an index into this array yielding a set of pointers (DP and LP). These point to the sector pointer array which inturn contains pointers to the shading data array.

Since with a very high probability all of the sectors in a subarea may not be on the screen, this data structure provides a convenient mechanism to skip over sectors which are not to be shaded either because the variable value is zero or the sector is not on the screen.

In order to maximize system performance the following form for the shading data was adopted. The shading data array contains the coordinates of the sector boundary line segments in counterclockwise

direction beginning with the endpoint with a minimum y (ordinate) coordinate. Furthermore this data is assumed to describe a polygonal body with no other sectors within its boundaries. If this is not the case and there is another sector within the polygonal boundaries, the containing sector must be partitioned into as many polygonal components as necessary so that the partitions contain no sectors within their boundaries. An entry for each of these polygons must be made in the sector pointer array using the same sector number. Since this is a rather rare phenomena this is not a severe restriction. For example the 290 sector MTC ZONE map contains only two sectors which are within the polygonal boundaries of another sector.

The current implementation restricts the number of sides for these polygonal shading definitions to be no more than 40. This has been found to be more than adequate in practical experience. If however a case arises where the number of sides exceeds 40 then either one of two courses of action is open. Either the sector can be partitioned into set of polygonal areas as described above or a simple programming modification can be made to increase the maximum number of sides permissible.

From the above it is easy to see that a sector may consist of two non-adjacent or contiguous polygonal areas. The definition of a map in section 1.2.1 is quite general in that it allows a sector to be the union of any finite set of polygonal areas.

3.3.2.4. The Subarea Alignment Data Structure

Figure 26 illustrates the subarea alignment data structure. The data in this structure gives the position of all subareas in a

map relative to contiguous subareas. Subareas are not to be confused with sectors. Refer to sections 2.3, 2.3.1, 2.3.2 and 3.3.1.

Subareas are 12 inch squares which cover the map like a checkerboard. Each subarea is given a positive integer value starting with one called the subarea number (SAN). Subareas are referenced by their subarea numbers. In order to display a window which straddles more than one subarea the connectivity of the subareas involved must be known. This information is contained in the subarea alignment data structure which permits the alignment of subareas. This data is also used by the MOVEX and MOVEY functions to move the window across the map.

ALDPT points to the subarea alignment data array. There are four cells in this array for every subarea of the map. The SAN of the subarea for which alignment data is desired is used as an index into this array. The first cell contains the SAN of the subarea directly above the indexing subarea and so forth as indicated in Figs. 26a and 26b. The absence of a subarea is denoted by the insertion of a negative number in the appropriate cell.

As previously noted at minimum magnification (.5) a maximum of nine subareas (corresponding to an area three subareas square) must be accessed to display a window. In the programming the subarea connectivity matrix (SACM(3,3)). This data structure allows for a single and efficient algorithm to set SACM and thus move the window across the map.

3.3.3. The Sector Name Data Structure

Figure 27 illustrates the sector name data structure. This structure contains the data necessary to output sector names on

the typewriter.

SNMPT the sector name disc record pointer points to the sector name array. The sector number is used as an index into this array to fetch a pair of pointers to the sector names pointer array. The first cell pointed to contains the number of names needed to completely identify the sector. This is followed by pointers to these names which are stored in the names array. The structure saves considerable space and redundancy. For example the full name for an MTC zone is MTC ZONE 5, SAN FRANCISCO COUNTY, CALIFORNIA. The number of names is three and only the number 5 need be changed for every MTC Zone in San Francisco county.

3.4. Transportation Files

As indicated in section 3.2.6, transportation maps are accessed using two indices: a map file number and a year. These two indices produce a transportation map file number, TFILN, which in turn is used by the program to access the appropriate data structure. If no transportation map exists TFILN is set = -1. Because at the moment there is no concerted interest in transportation studies by the people in the Urban Systems Group and because there is a strong desire to exploit the existing system before proceeding to other activities this phase of BUDS has not been emphasized. Careful consideration of transportation study objectives will have a strong influence on further software implementation in the transportation realm. This in turn may necessitate a revision of the transportation file structure. Since one of the design objectives

of BUDS was to facilitate any implementation changes the current implementation itself will impose few, if any, restrictions.

3.4.1. The Transportation File Structure

The transportation file structure is illustrated in Fig. 28. TFILN the transportation file number points to an array containing three disc record pointers TGMPT, the transportation global map disc record pointer, TSAPT, the transportation subarea disc record pointer and TRNPT, the transportation name disc record pointer.

TGMPT points to the transportation global map data array which contains the data necessary to overlay a transportation grid on the global map described in sections 2.3, 2.3.3 and 3.3.1. This array contains entries in groups of three cells. The first cell contains the type of graphic instruction, T, to carry out on the coordinates contained in the next two cells. For example $T_i = 2$ is a command to draw a line from he current beam position to the coordinate (x_i, y_i). See Fig. 28 for the other allowable graphic instruction types. Since there is only one entity, T = 4 is a forbidden type for the global map.

TSAPT points to the subarea disc record pointer array. The subarea number of the desired subarea is used as an index into this array to access a pair of pointers. These pointers are used to access the subarea transportation data array which is indexed by a subarea number. The latter structure contains the necessary data to construct the transportation facilities map subarea by subarea. A mechanism similiar to that described in section 3.3.2.4 allows transportation facilities to be displayed over the map.[7]

[7] The subarea connectivity matrix SACM is used to select the proper subareas.

The first word pointed to in the subarea transportation data array is the transportation number of the facility to be constructed. Transportation facilities are referenced by their transportation facility number, TRN. For identification purposes transportation facilities are treated like sectors in the map however their entity values are negative to distinguish them from sectors. Thus a transportation facility with number TRN will have an entity value - TRN when displayed on the screen. The data following TRN in this structure is exactly like that in the global transportation map data array except for the following. T = 4 signals the beginning of a new transporation facility definition.

The transportation name identification structure is exactly the same as one described in section 3.3.3 for sectors. Therefore it is not discussed here.

CHAPTER 4

CONCLUDING REMARKS

4.1. Concluding Remarks

Empirical studies of urban systems are impeded by the complexity of urban systems and the nature of urban data. In particular the large number of variables implies large data bases. Secondly three attributes characterize urban variables, quantity or quality, location and time. This thesis has attempted to describe an interactive computer graphics urban information system which presents urban data in a highly intelligible or comprehensible form which takes these attributes into account.

The capabilities of man-machine interactive systems such as BUDS are difficult to describe with words. They are more fully conveyed through system operation. Interest both within and outside the university has been generated through word-of-month which lead to several system demonstrations. During these demonstrations it was noted that the system response to each query usually evoked a series of "follow-up" queries which were answered immediately. It was also noted that in one instance misconceptions concerning the population distribution within San Francisco were dispelled.

BUDS is an evolving system and software development is now at the point where many empirical studies can be performed easily. The first studies will probably involve statistical testing of hypotheses of urban development in which certain characteristics of old communities will be compared to those of new communities in the San Francisco Bay Area.

As far as future developments are concerned, certain graph drawing capabilities such as histograms, Engel's Curve, etc. are in the final stages of development. The next step should be the development of an automatic facility to add and delete data. Incorporated in this capability should be the ability to add user generated data (from created variables) to the data bank. Although there are many other possibilities experience with the system is needed to determine which of these should be implemented and with what priority.

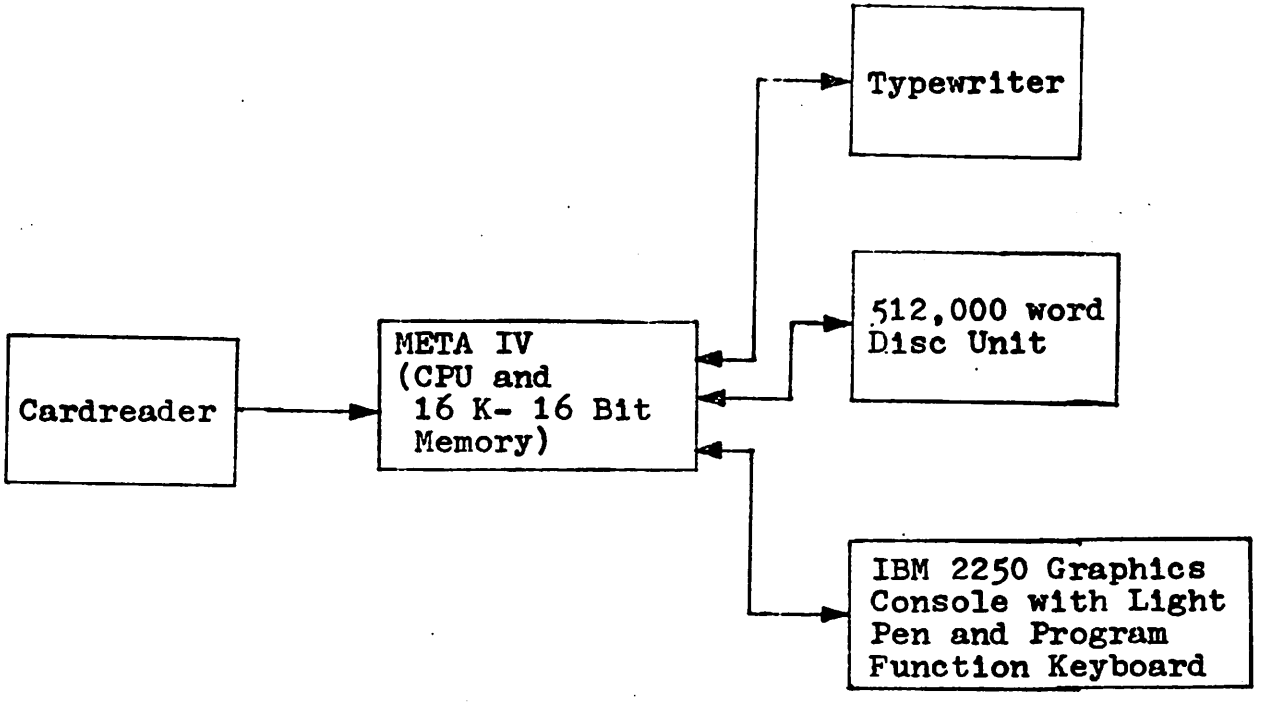


Fig. 1 The Computer Equipment

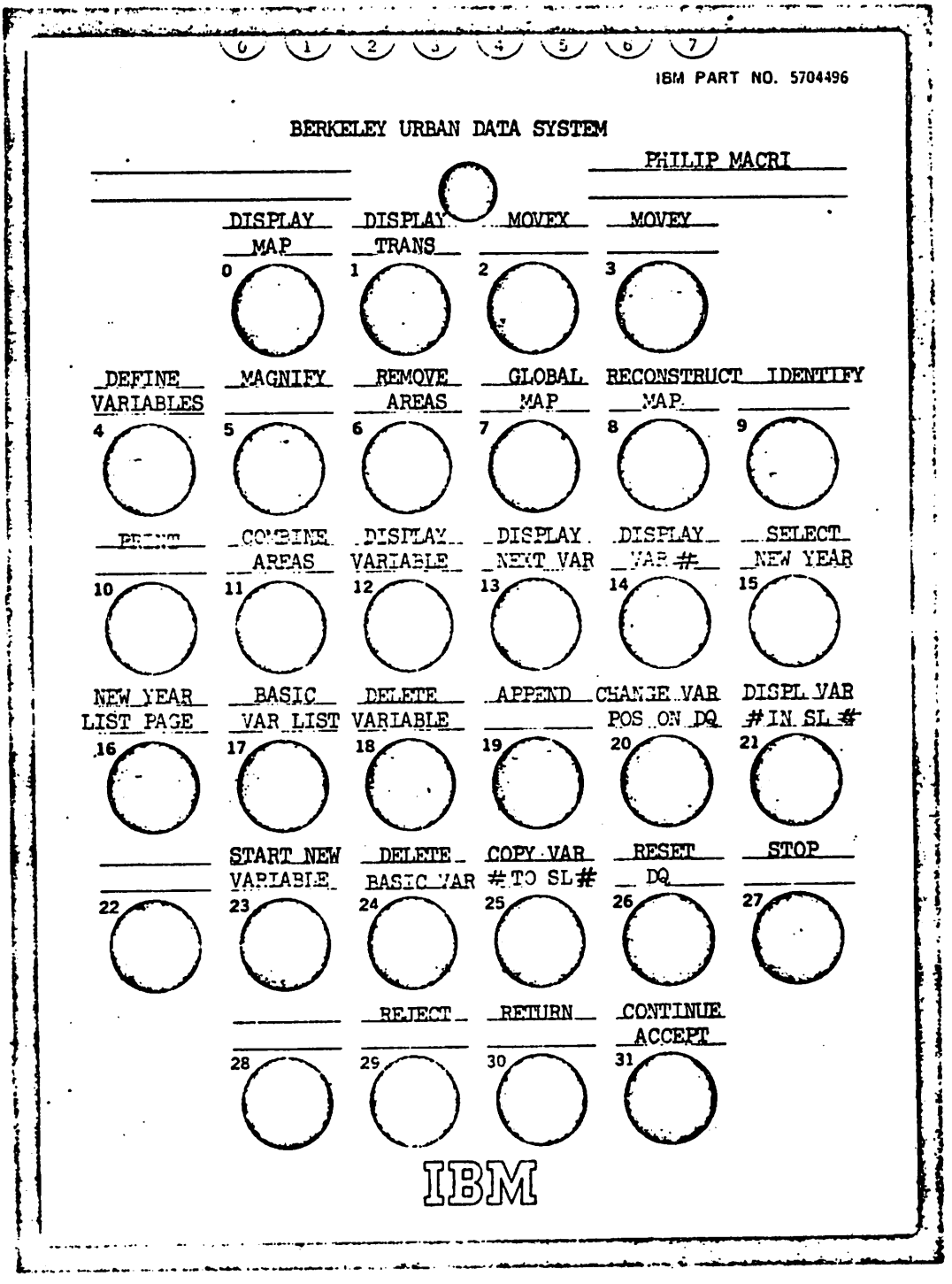


Fig. 2a The Program Function Keyboard

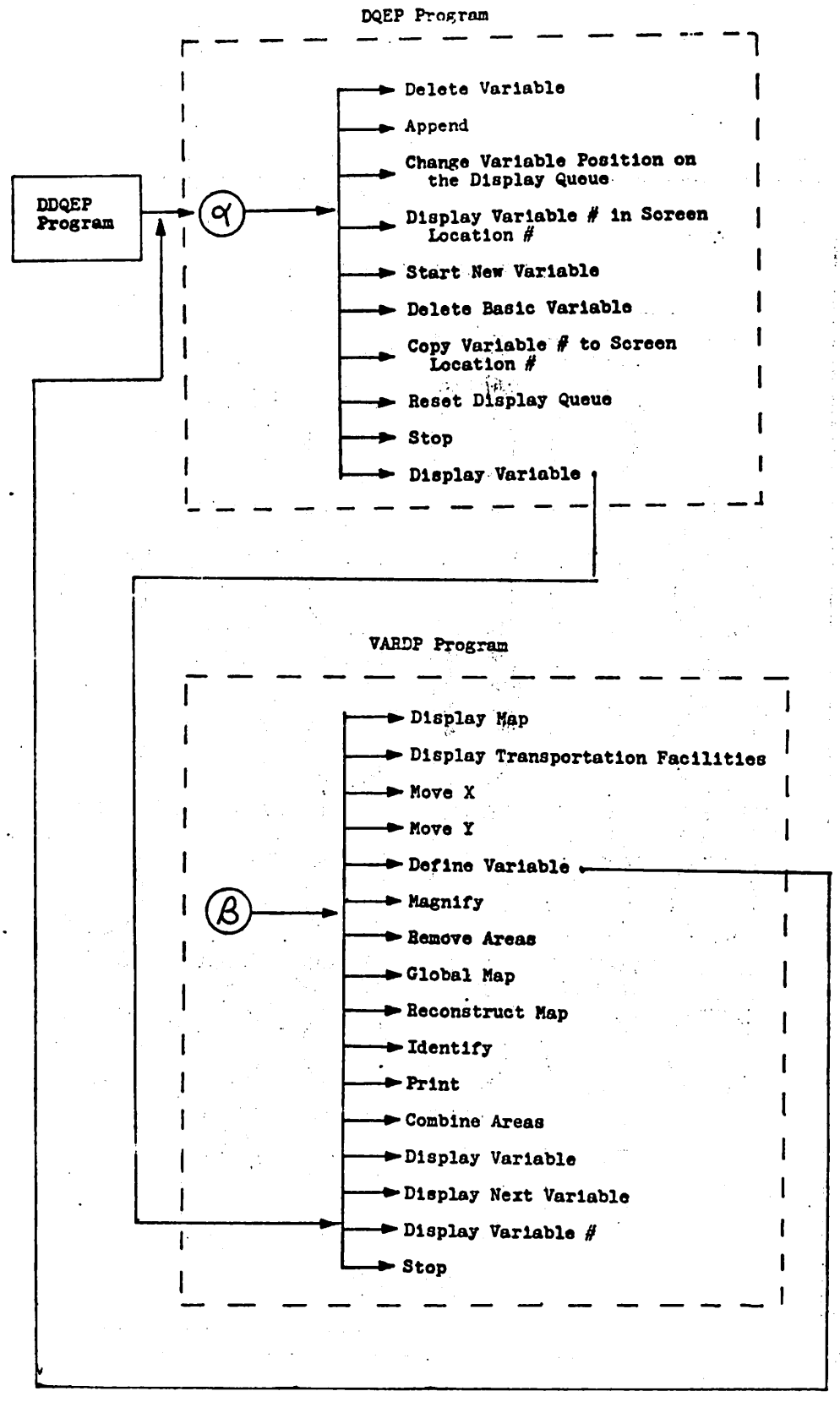


Fig. 2b-1 Function Accessibility

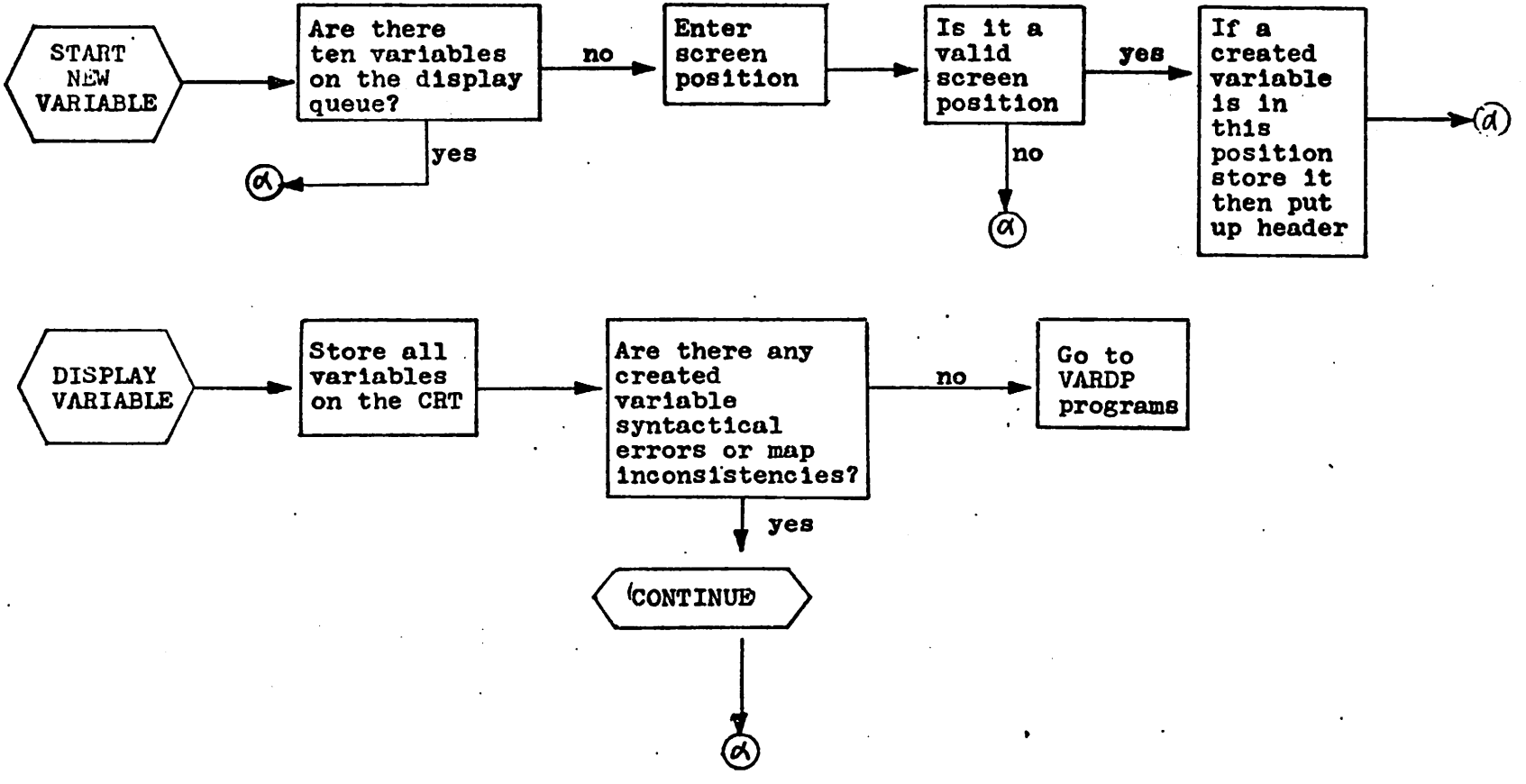

START NEW VARIABLE
Are there ten variables on the display queue?
no
yes
Enter screen position
Is it a valid screen position
yes
no
If a created variable is in this position store it then put up header
DISPLAY VARIABLE
Store all variables on the CRT
Are there any created variable syntactical errors or map inconsistencies?
no
yes
Go to VARDP programs
CONTINUE

Fig. 2b-2

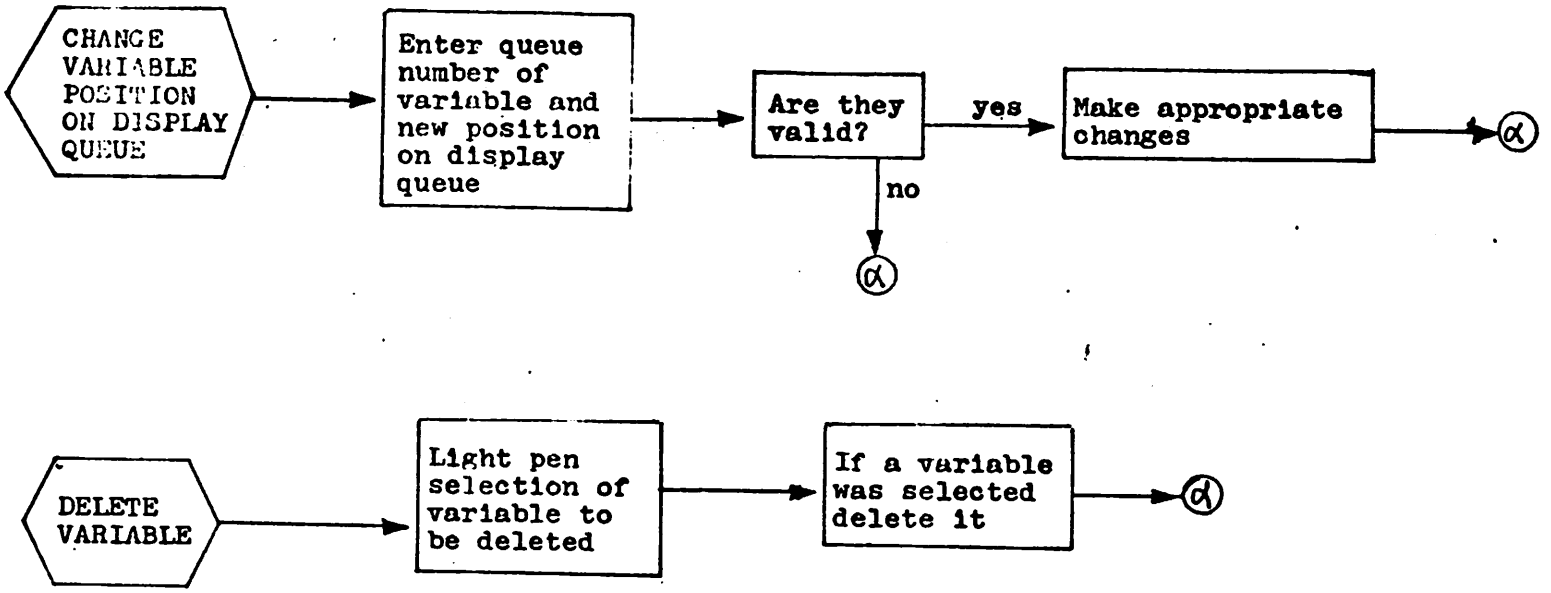

CHANGE VARIABLE POSITION ON DISPLAY QUEUE
Enter queue number of variable and new position on display queue
Are they valid?
yes
Make appropriate changes
α
no
α
DELETE VARIABLE
Light pen selection of variable to be deleted
If a variable was selected delete it
α

Fig. 2b-3

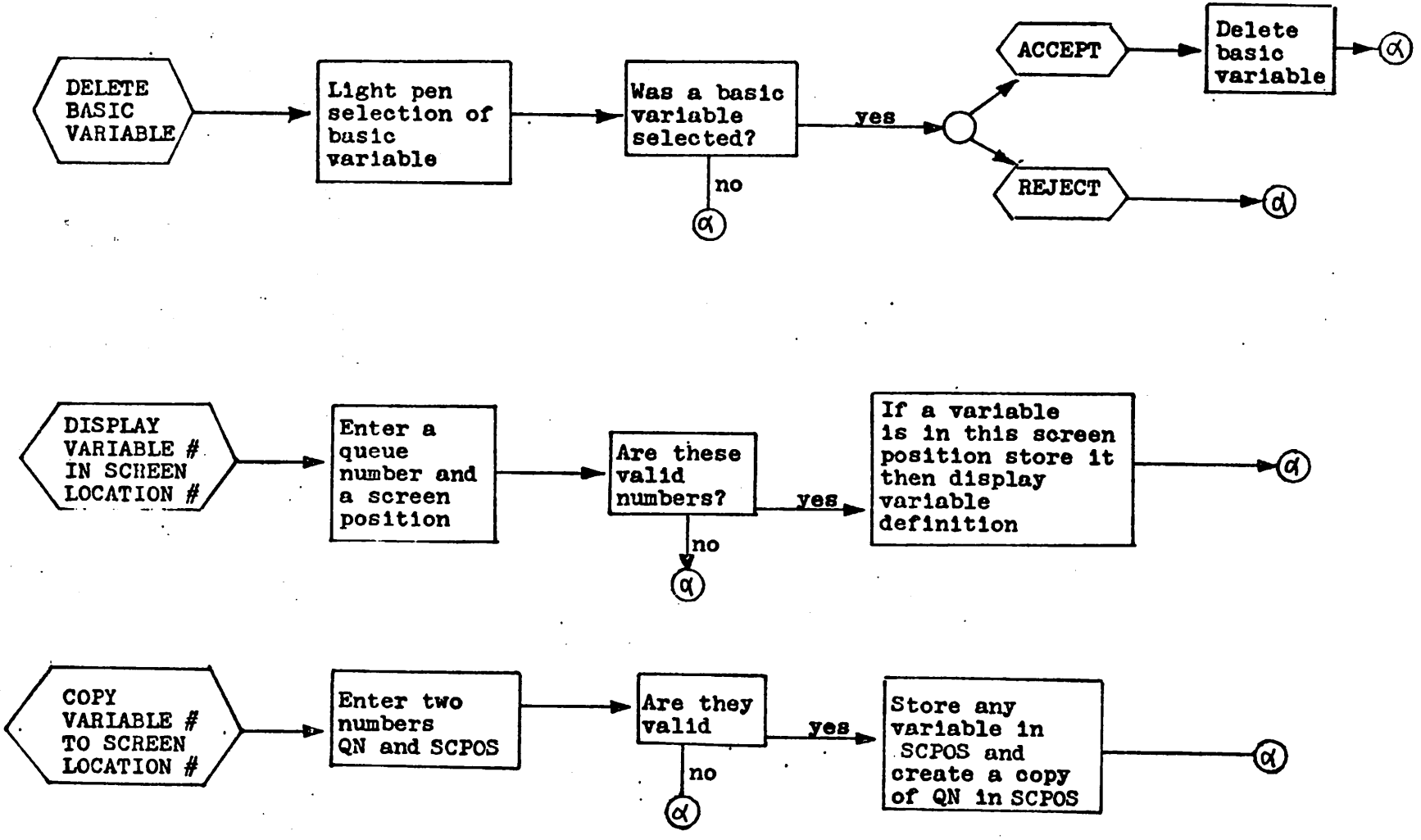

DELETE BASIC VARIABLE
Light pen selection of basic variable
Was a basic variable selected?
yes
no
α
ACCEPT
Delete basic variable
REJECT
DISPLAY VARIABLE # IN SCREEN LOCATION #
Enter a queue number and a screen position
Are these valid numbers?
yes
no
If a variable is in this screen position store it then display variable definition
COPY VARIABLE # TO SCREEN LOCATION #
Enter two numbers QN and SCPOS
Are they valid
yes
no
Store any variable in SCPOS and create a copy of QN in SCPOS

Fig. 2b-4

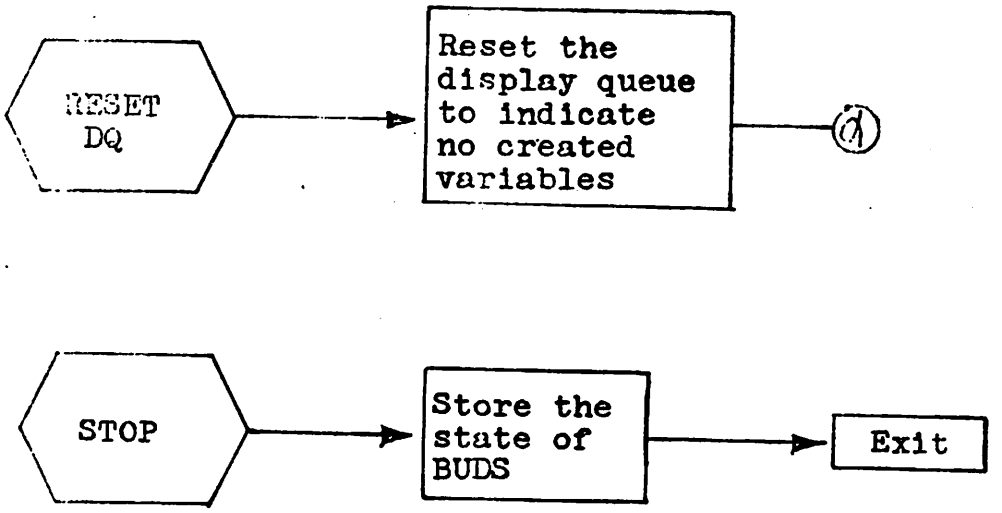

RESET
DQ
Reset the
display queue
to indicate
no created
variables
d
STOP
Store the
state of
BUDS
Exit

Fig. 2b-5

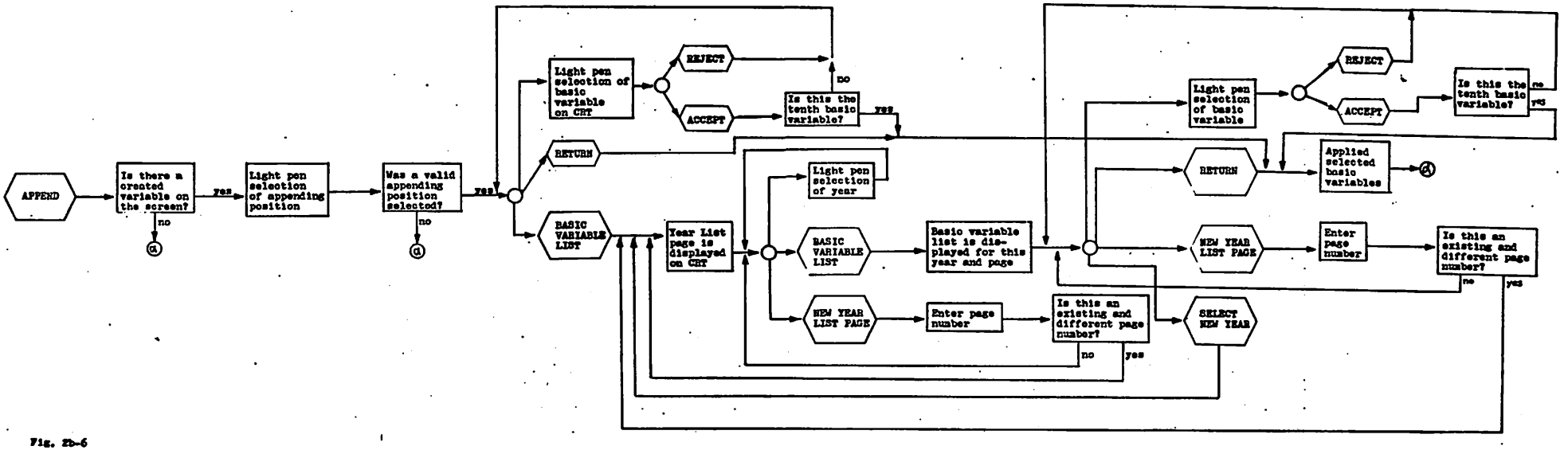

APPEND
Is there a created variable on the screen?
yes
no
Light pen selection of appending position
Was a valid appending position selected?
yes
no
Light pen selection of basic variable on CRT
REJECT
ACCEPT
Is this the tenth basic variable?
no
yes
RETURN
BASIC VARIABLE LIST
Year List page is displayed on CRT
Light pen selection of year
BASIC VARIABLE LIST
Basic variable list is displayed for this year and page
NEW YEAR LIST PAGE
Enter page number
Is this an existing and different page number?
no
yes
SELECT NEW YEAR
Light pen selection of basic variable
REJECT
ACCEPT
Is this the tenth basic variable?
no
yes
RETURN
Applied selected basic variables
NEW YEAR LIST PAGE
Enter page number
Is this an existing and different page number?
no
yes

Fig. 2b-6

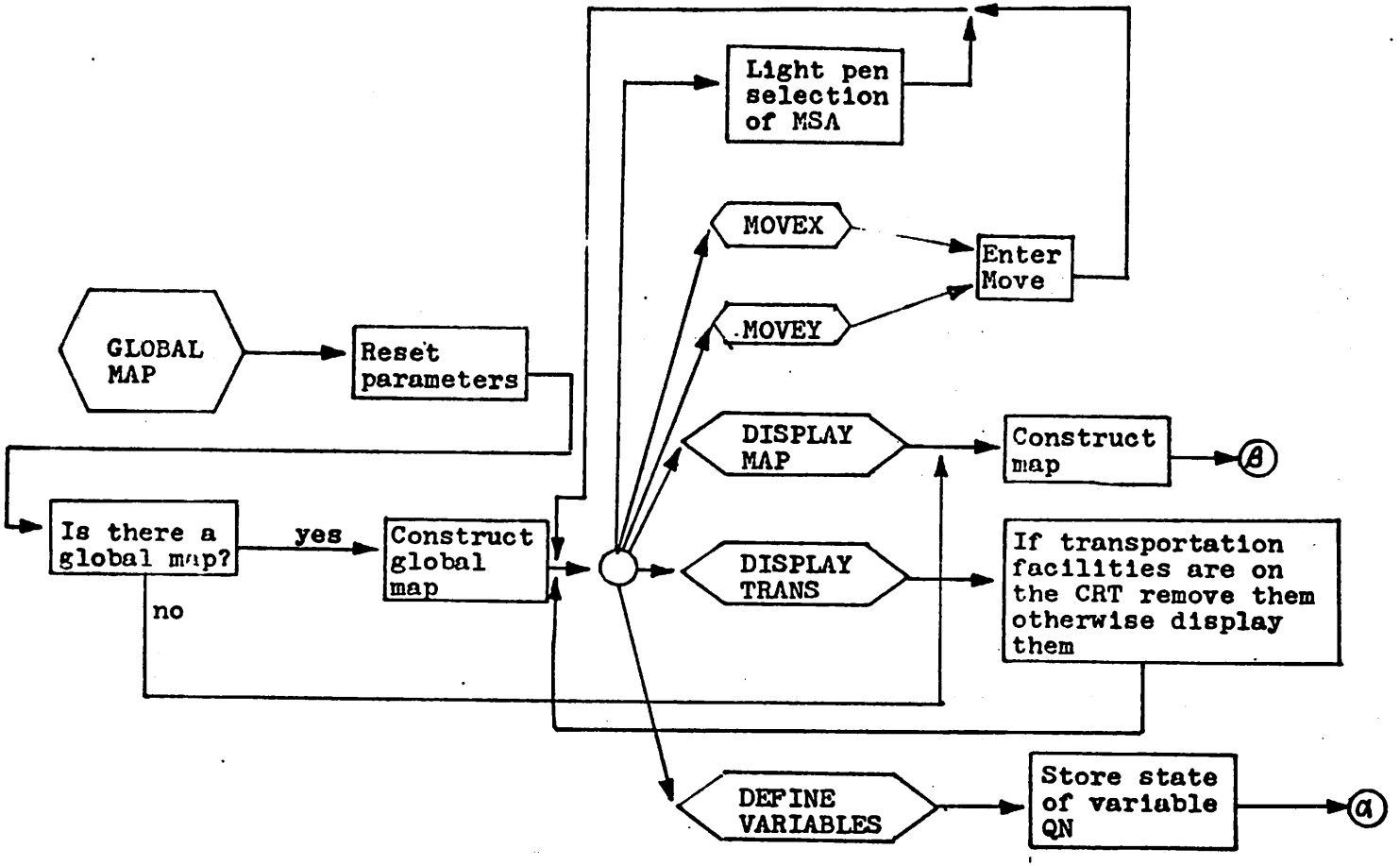

GLOBAL MAP
Reset parameters
Is there a global map?
yes
no
Construct global map
Light pen selection of MSA
MOVEX
MOVEY
Enter Move
DISPLAY MAP
Construct map
β
DISPLAY TRANS
If transportation facilities are on the CRT remove them otherwise display them
DEFINE VARIABLES
Store state of variable QN
α

Fig. 2b-7

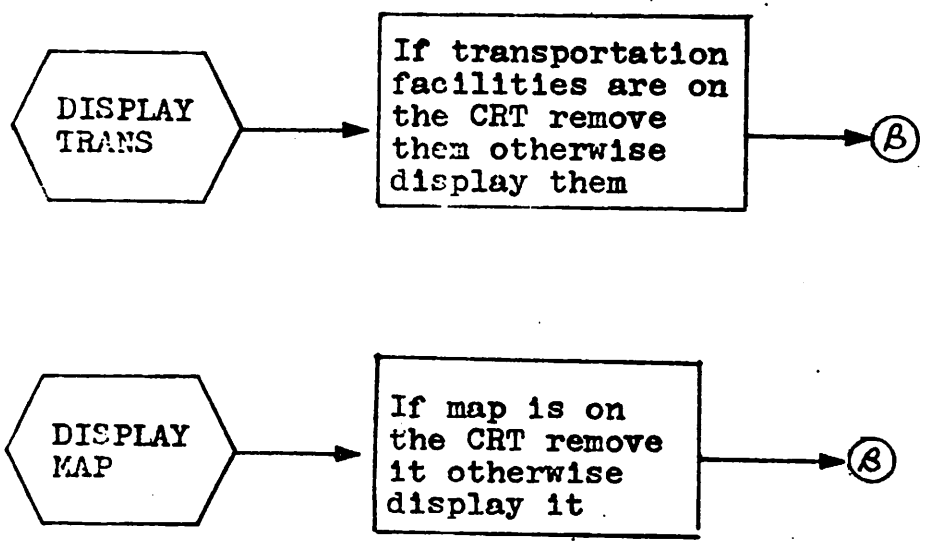

DISPLAY TRANS
If transportation facilities are on the CRT remove them otherwise display them
B
DISPLAY MAP
If map is on the CRT remove it otherwise display it
B

Fig. 2b-8

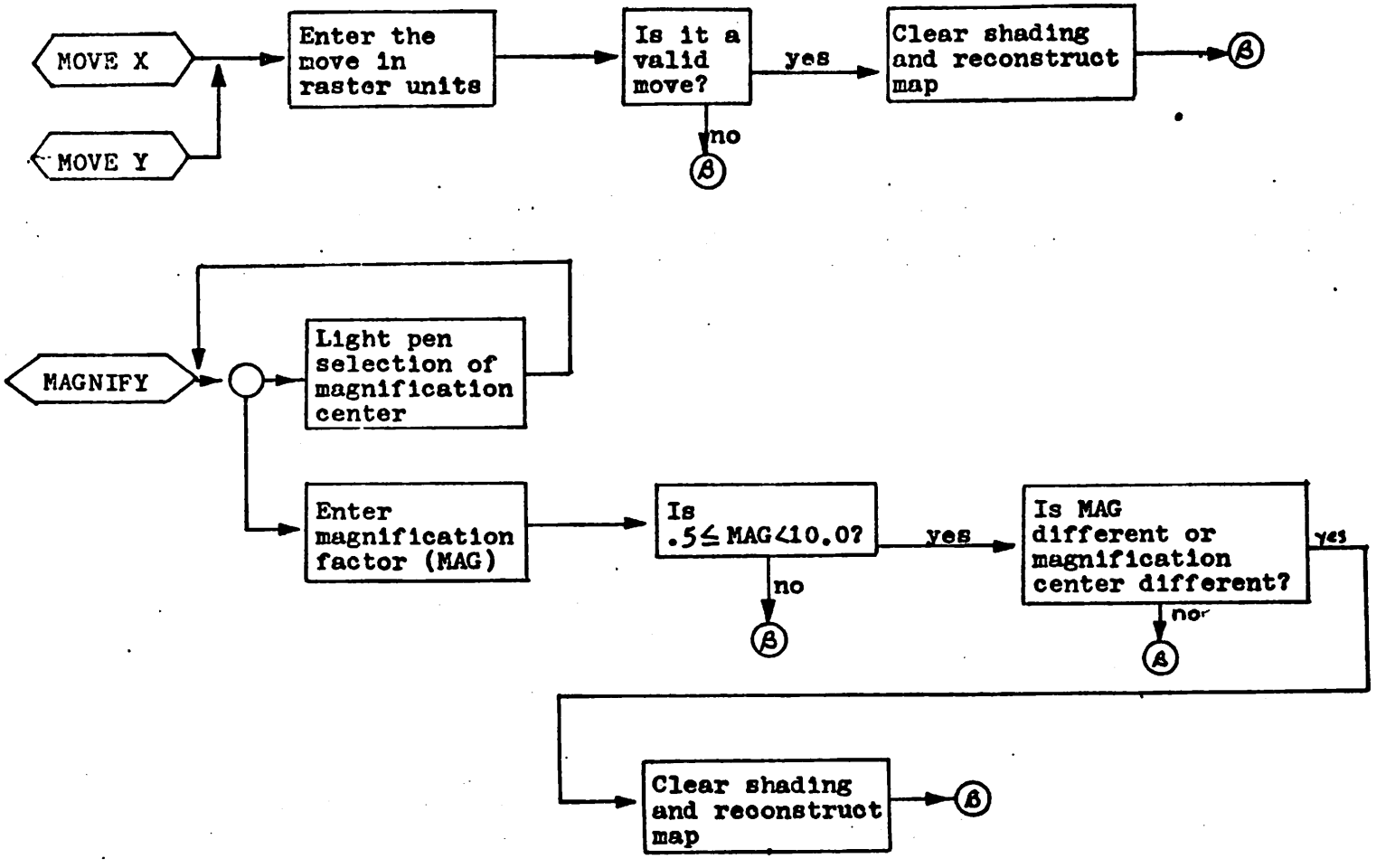

MOVE X
MOVE Y
Enter the move in raster units
Is it a valid move?
yes
no
Clear shading and reconstruct map
MAGNIFY
Light pen selection of magnification center
Enter magnification factor (MAG)
Is .5≤ MAG<10.0?
yes
no
Is MAG different or magnification center different?
yes
no
Clear shading and reconstruct map

Fig. 2b-9

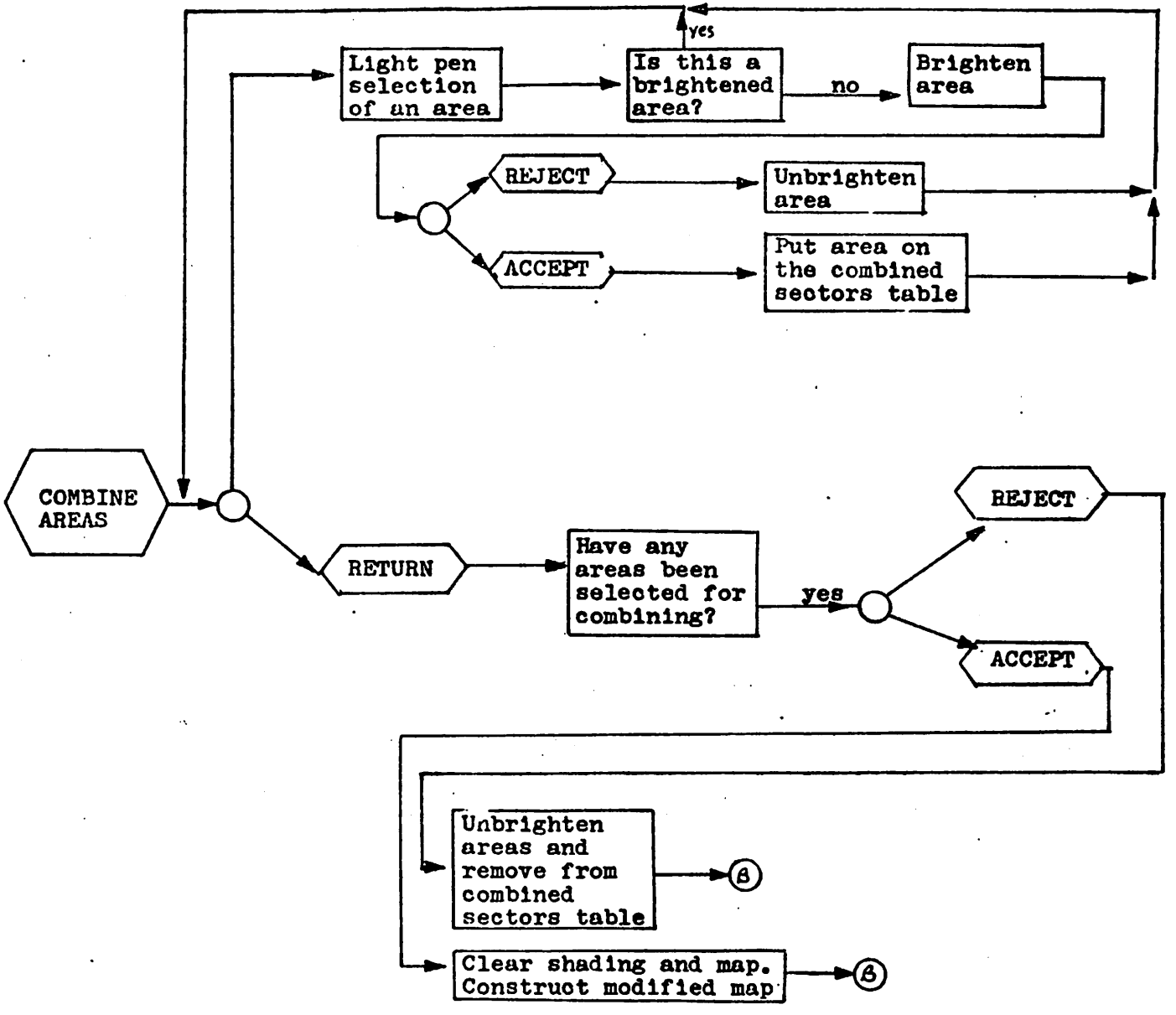

yes
Light pen selection of an area
Is this a brightened area?
no
Brighten area
REJECT
Unbrighten area
ACCEPT
Put area on the combined sectors table
COMBINE AREAS
RETURN
Have any areas been selected for combining?
yes
REJECT
ACCEPT
Unbrighten areas and remove from combined sectors table
B
Clear shading and map. Construct modified map
B

Fig. 2b-10

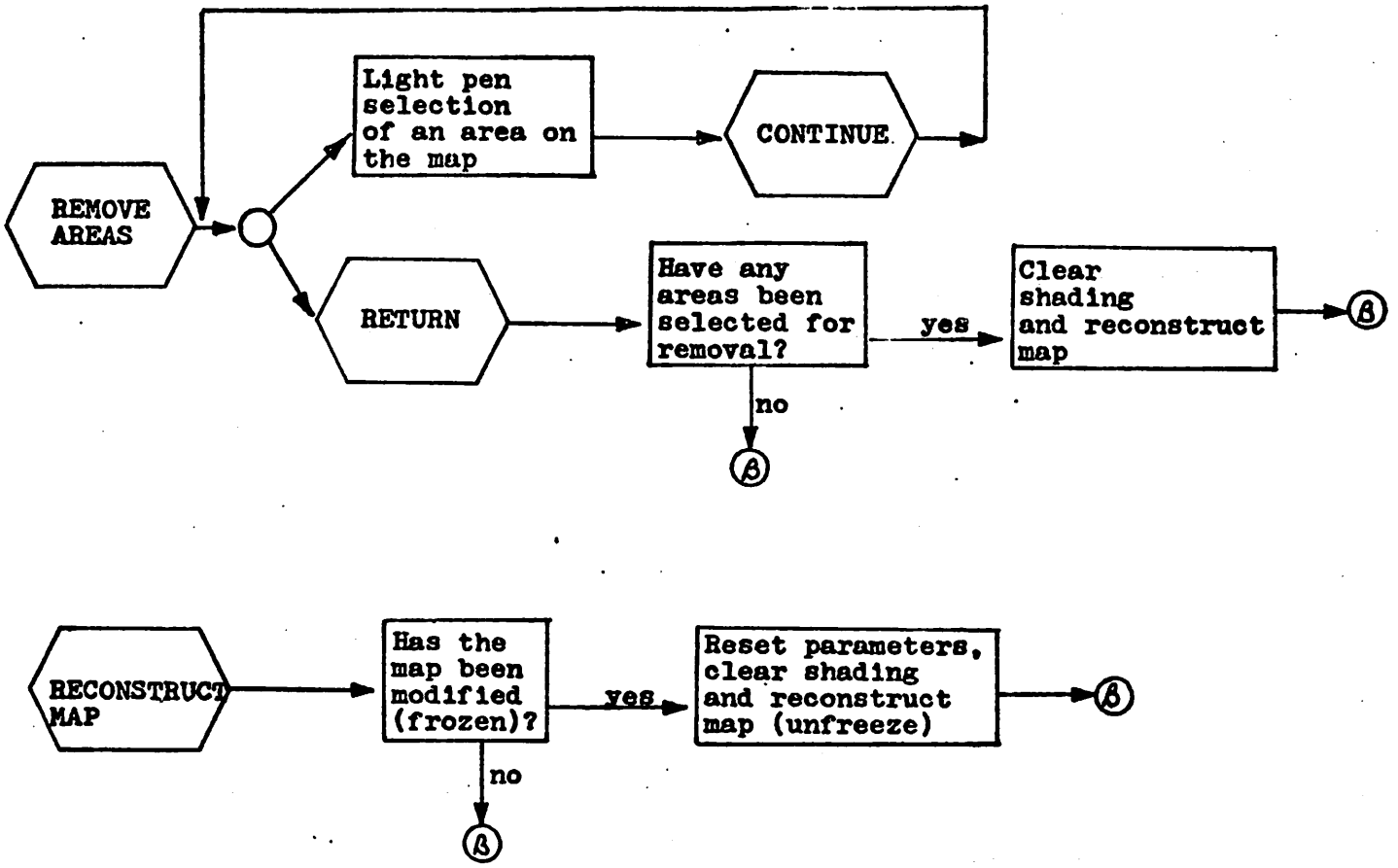

REMOVE AREAS
Light pen selection of an area on the map
CONTINUE
RETURN
Have any areas been selected for removal?
yes
Clear shading and reconstruct map
A
no
A
RECONSTRUCT MAP
Has the map been modified (frozen)?
yes
Reset parameters, clear shading and reconstruct map (unfreeze)
A
no
A

Fig. 2b-11

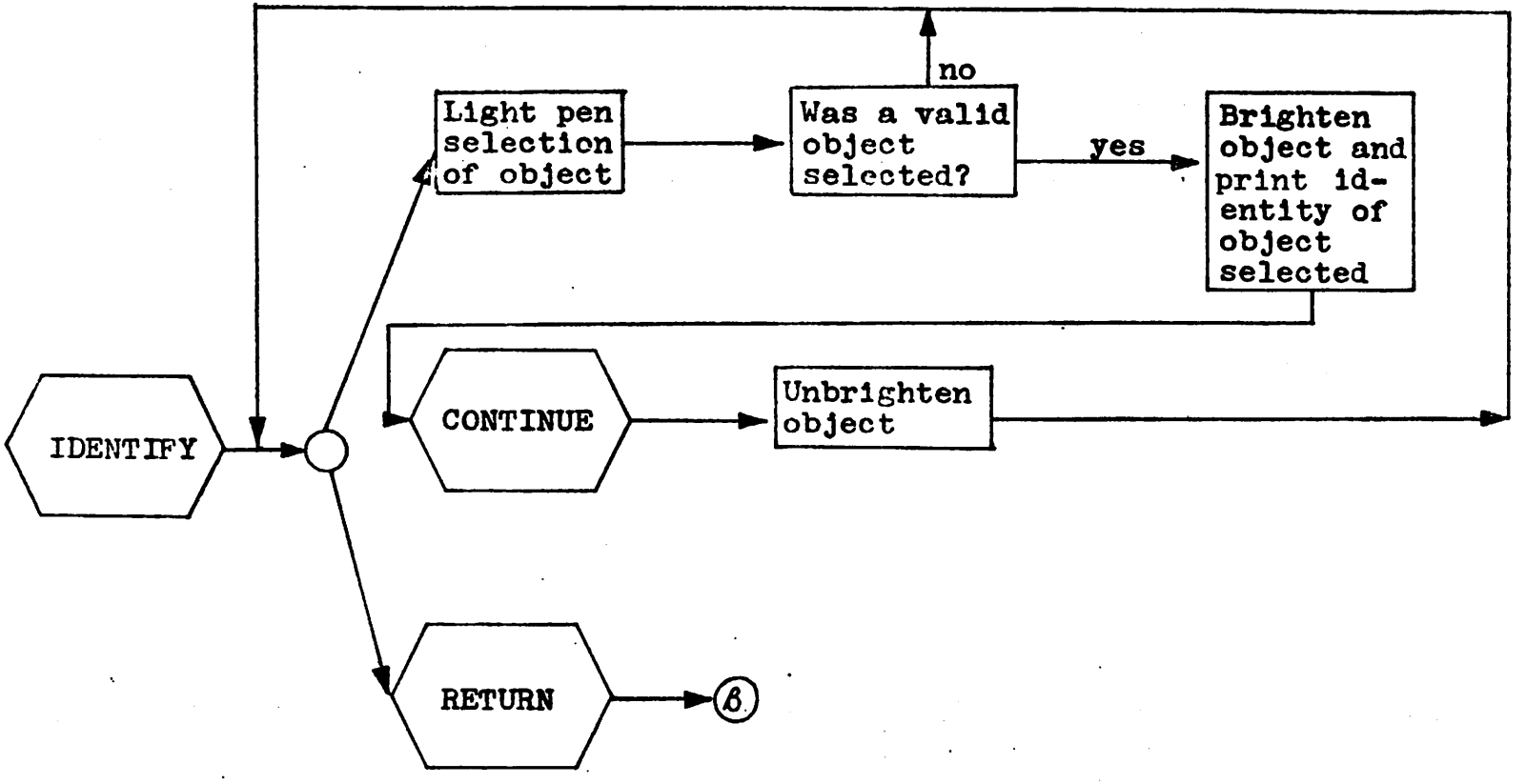

IDENTIFY
Light pen selection of object
Was a valid object selected?
no
yes
Brighten object and print identity of object selected
CONTINUE
Unbrighten object
RETURN
B

Fig. 2b-12

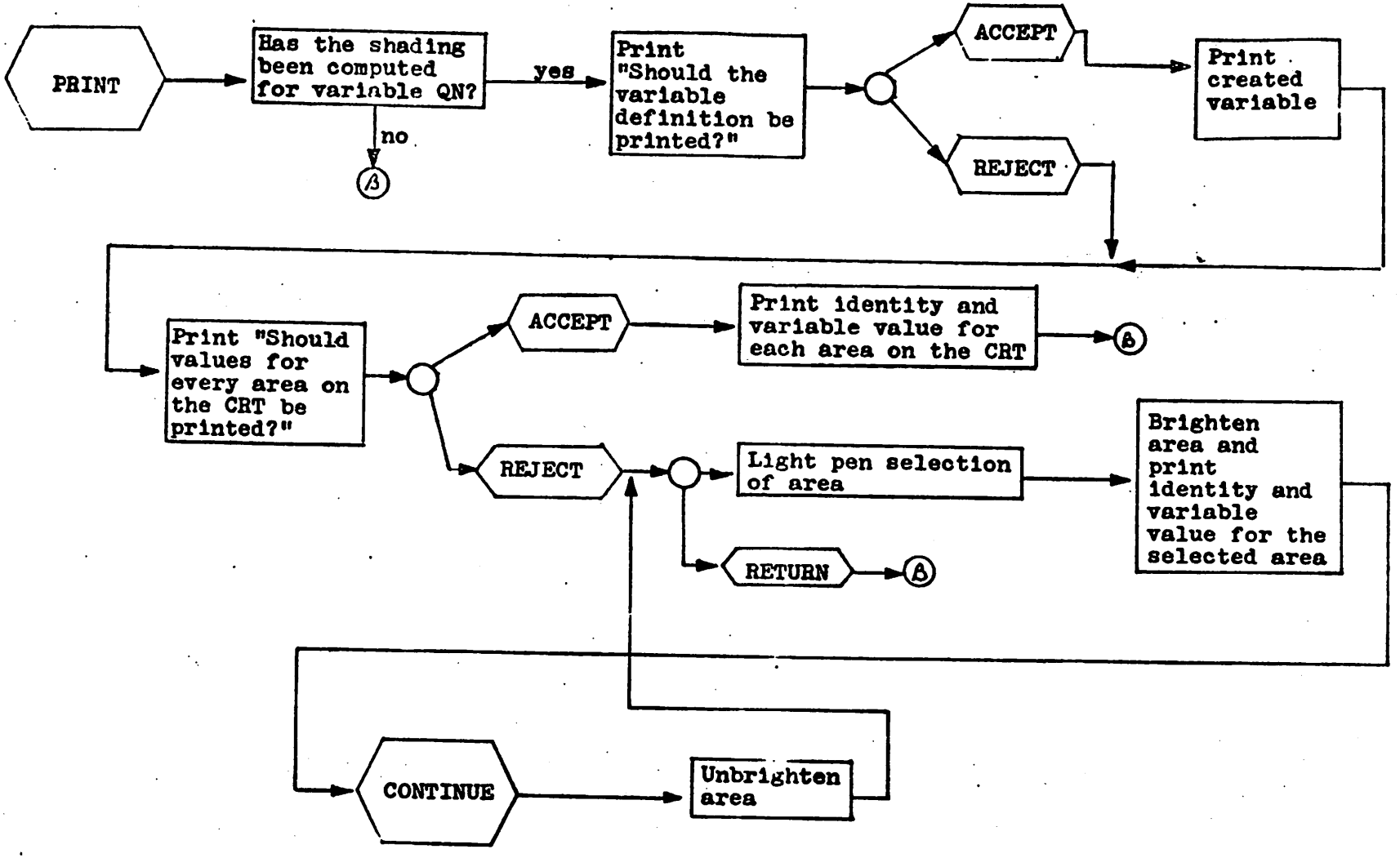

PRINT
Has the shading been computed for variable QN?
yes
no
Print "Should the variable definition be printed?"
ACCEPT
REJECT
Print created variable
Print "Should values for every area on the CRT be printed?"
ACCEPT
Print identity and variable value for each area on the CRT
REJECT
Light pen selection of area
RETURN
Brighten area and print identity and variable value for the selected area
CONTINUE
Unbrighten area

Fig. 2b-13

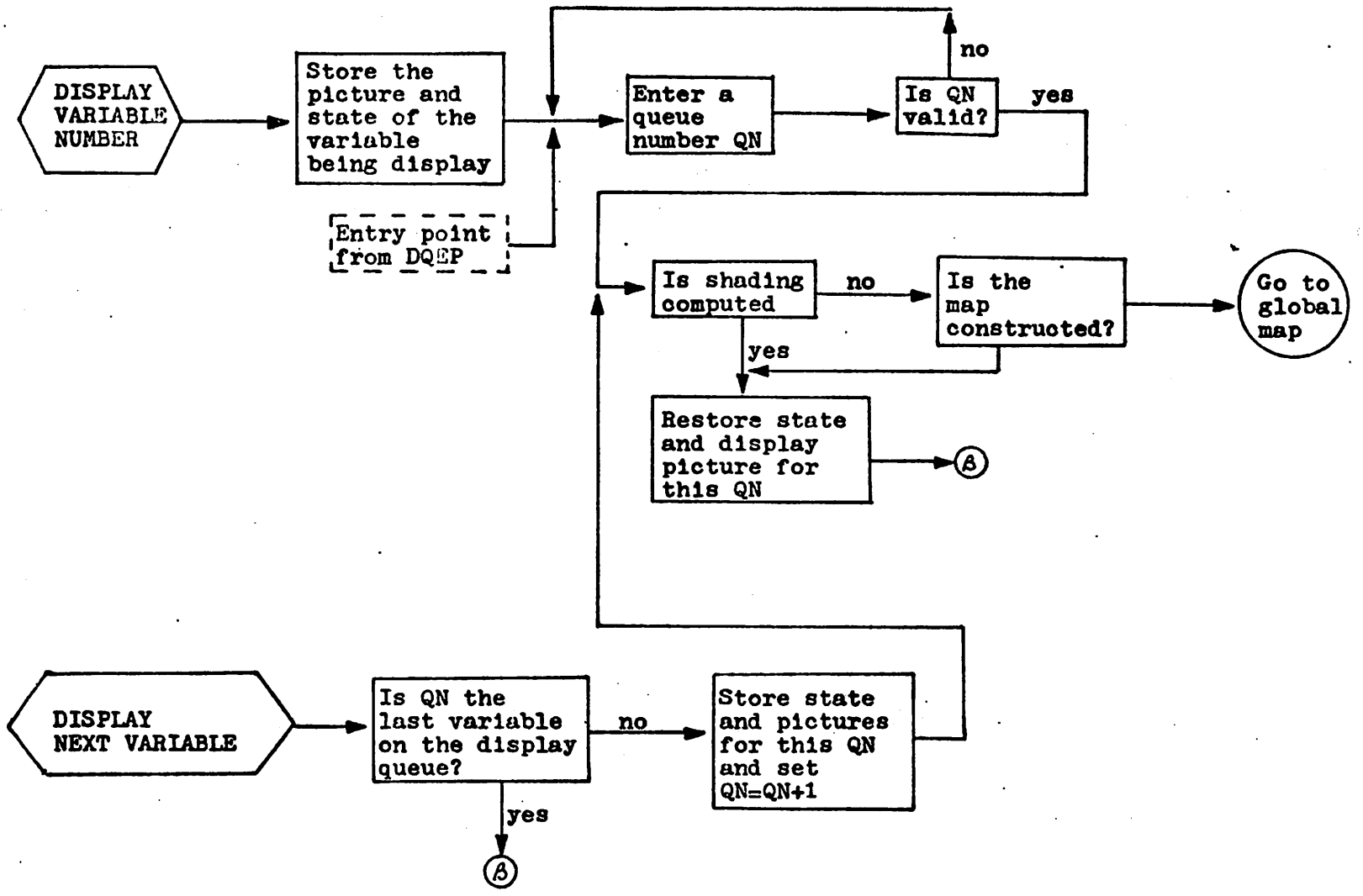

DISPLAY VARIABLE NUMBER
Store the picture and state of the variable being display
Entry point from DQEP
Enter a queue number QN
Is QN valid?
no
yes
Is shading computed
no
Is the map constructed?
Go to global map
yes
Restore state and display picture for this QN
β
DISPLAY NEXT VARIABLE
Is QN the last variable on the display queue?
no
Store state and pictures for this QN and set QN=QN+1
yes
β

Fig. 2b-14

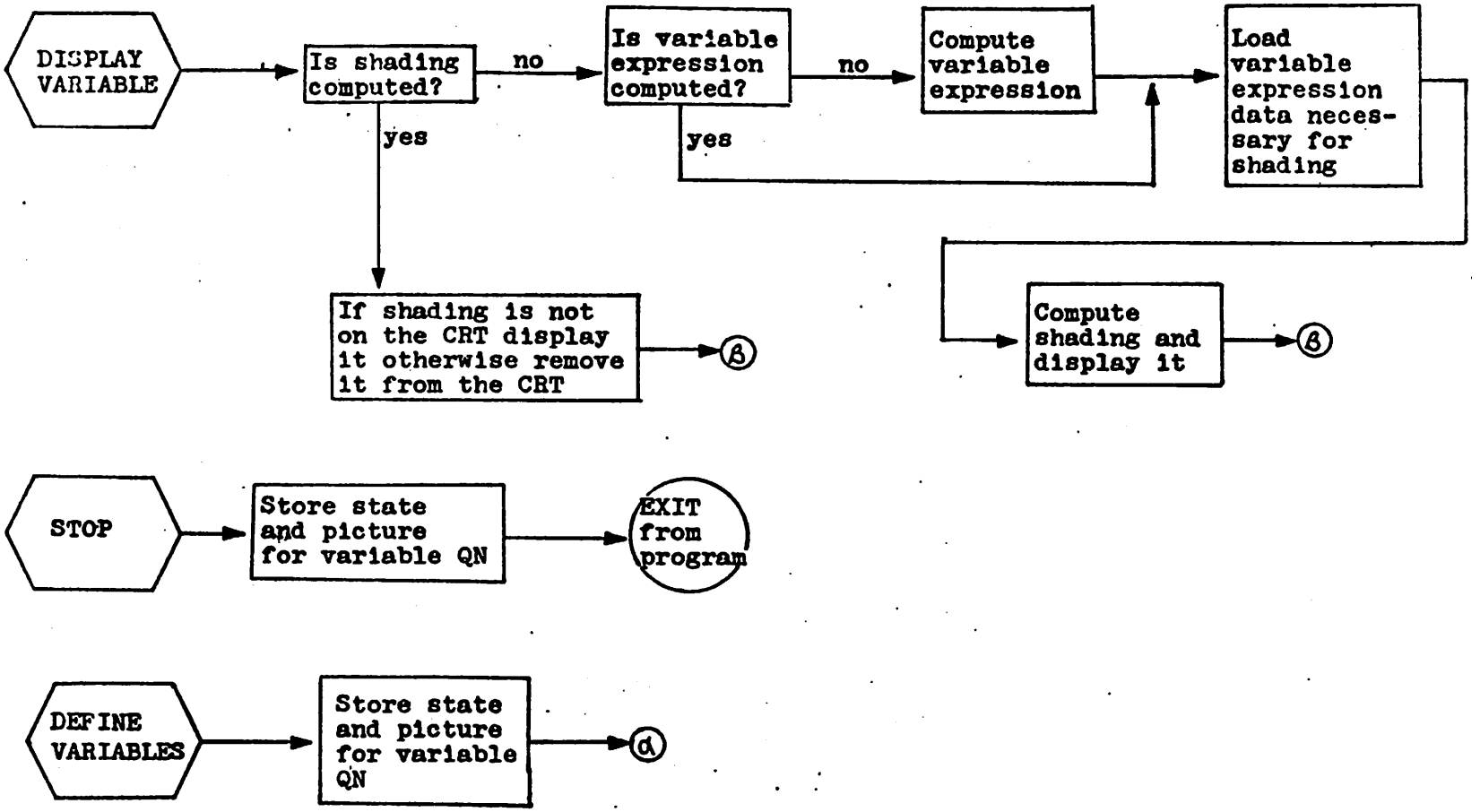

DISPLAY VARIABLE
Is shading computed?
no
Is variable expression computed?
no
Compute variable expression
Load variable expression data necessary for shading
yes
yes
If shading is not on the CRT display it otherwise remove it from the CRT
Compute shading and display it
STOP
Store state and picture for variable QN
EXIT from program
DEFINE VARIABLES
Store state and picture for variable QN

Fig. 2b-15

QN= 1 NEW NORMALIZATION CONSTANT

A- POPULATION, TOTAL, MTC ZN, 1970
B- RES POP (INCL RURAL), TOTAL, MTC ZN, 1960

Fig. 3 A Created Variable.

QN= 1 NEW NORMALIZATION CONSTANT

Fig. 4a Header created by selecting the START NEW VARIABLE function.

QN= 1 SAME NORMALIZATION CONSTANT

Fig. 4b Header as a result of touching NEW of Fig. 4a with the light pen and accepting with the ACCEPT function.

QN= 1 LOW-HIGH SHADING
X +0.0000E+00 X +0.0000E+00 X

Fig. 4c Header as a result of touching NORMALIZATION CONSTANT of Fig. 4a or Fig. 4b with the light pen and accepting with the ACCEPT function.

Fig. 4 Changing the Variable Header.

QN= 03 LOW-HIGH SHADING
L +9.4799E+00 H +7.1999E+01 H
A- POPULATION, TOTAL, MTC ZN, 1970
B- TOTAL ACRES, MTC ZN, 1970
EXP= A/B

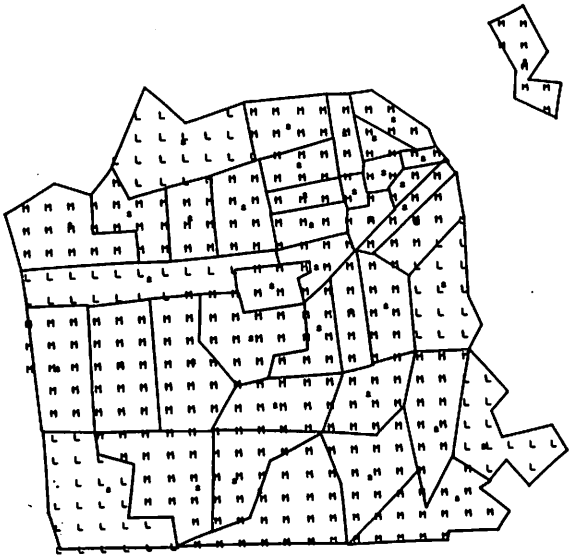


Fig. 5 A LOW-HIGH Query: definition and answer.

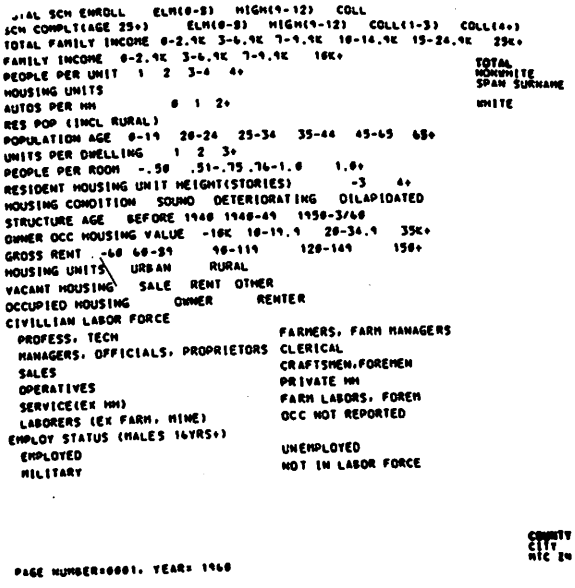


Fig. 6 A Page of Basic Variables. The map list (COUNTY, etc.), list of secondary attributes (TOTAL, etc.) and list of primary attributes (0-19, etc.) modify the urban name POPULATION AGE.

GN= 84 NEW NORMALIZATION CONSTANT
A- FAMILY INCOME, 25K+, TOTAL, COUNTY, 1976
B- POPULATION, TOTAL, COUNTY, 1976
EXP= A/B

Fig. 7a A Created Variable

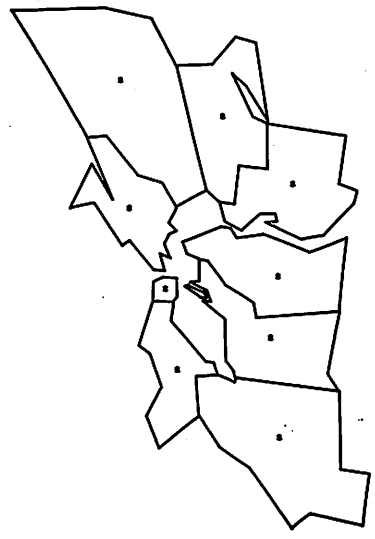


Fig. 7b The Map of Fig. 7a: The Nine County Bay Area.

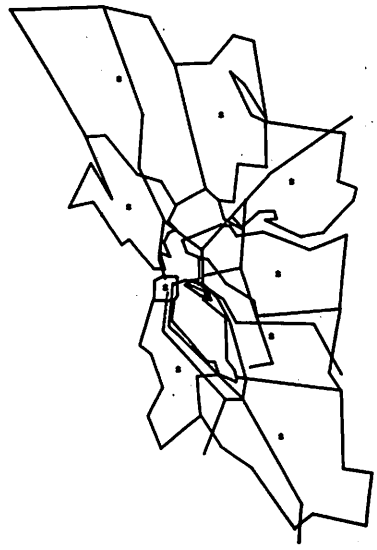


Fig. 7c The Map of Fig. 7a with Transportation Facilities.

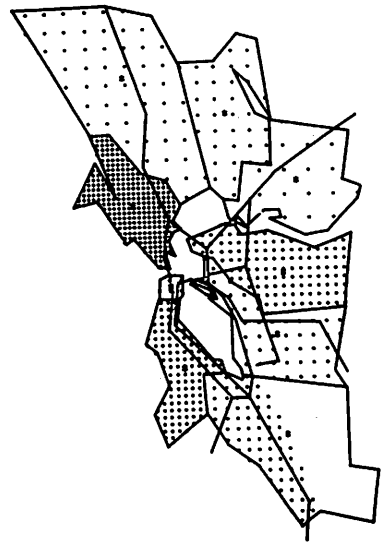


Fig. 7: Response to the Query of Fig. 7a.

```
QN= 07          NEW NORMALIZATION CONSTANT
A- FAMILY INCOME, 0-2.9K, TOTAL, MTC ZN, 1970
B- FAMILY INCOME, 0-2.9K, TOTAL, CITY, 1970
EXP= 100.0*(A/B)
```

Fig. 8a A Created Variable with Two Different Maps

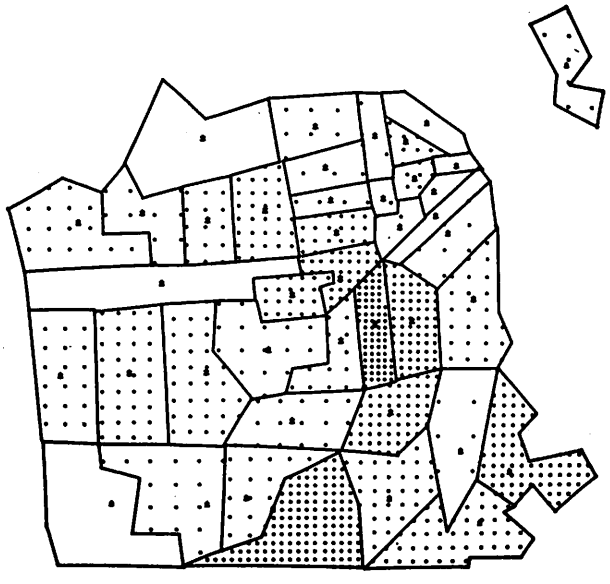


Fig. 8b Response to Query of Fig. 8a. MTC zone is the Display Map.

Fig. 8 Mixing Maps in a Created Variable

```
QN= 02          NEW NORMALIZATION CONSTANT
A- FAMILY INCOME, 0-2.9K, NONWHITE, MTC ZN, 1960
EXP= A
```

```
QN= 03          SAME NORMALIZATION CONSTANT
A- FAMILY INCOME, 0-2.9K, TOTAL, MTC ZN, 1970
B- FAMILY INCOME, 0-2.9K, WHITE, MTC ZN, 1970
EXP= A-B
```

Fig. 9a Two created variables. Each represents the distribution of poor nonwhite families.

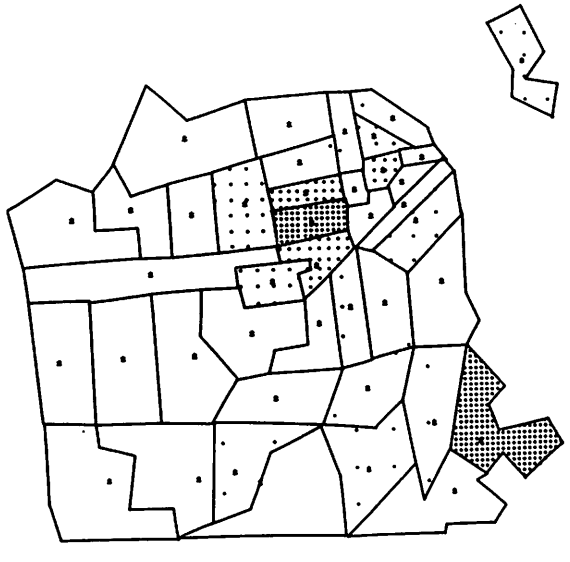


Fig. 9b The response to the variable with QN= 2 in Fig. 9a.

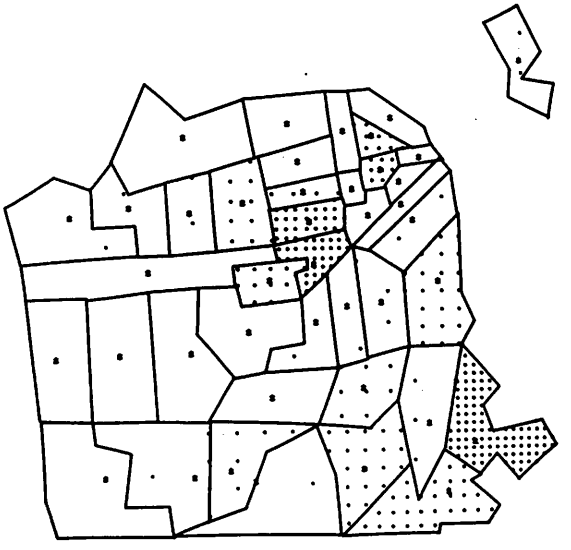


Fig. 9c The response to the variable with QN= 3 in Fig. 9a.

Fig. 9 An Intertemporal Comparison. Fig. 9c shows a spreading out of poor nonwhite families as compared to Fig. 9b.

```
QN= 08          NEW NORMALIZATION CONSTANT
A- RES POP (INCL RURAL), NONWHITE, MTC ZN, 1960
B- RES POP (INCL RURAL), BLACK, MTC ZN, 1960
C- TOTAL ACRES, MTC ZN, 1970
ERP= (A-B)/C
```

Fig. 10a A created variable asking for the oriental population density in San Francisco in 1960.

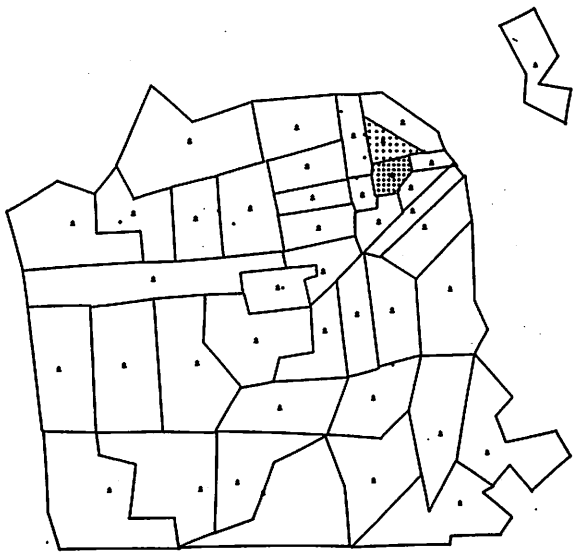


Fig. 10b The repense to the query of Fig. 10a.

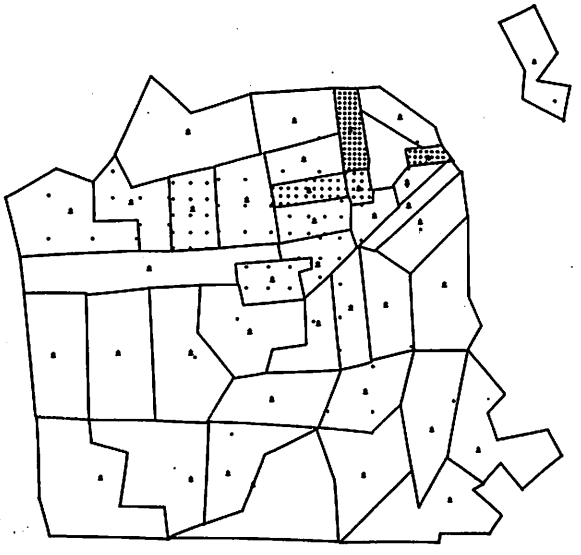


Fig. 10c The response to the query of Fig. 10a after the removal of the two most densely populated areas.

Fig. 10 Removing Sectors

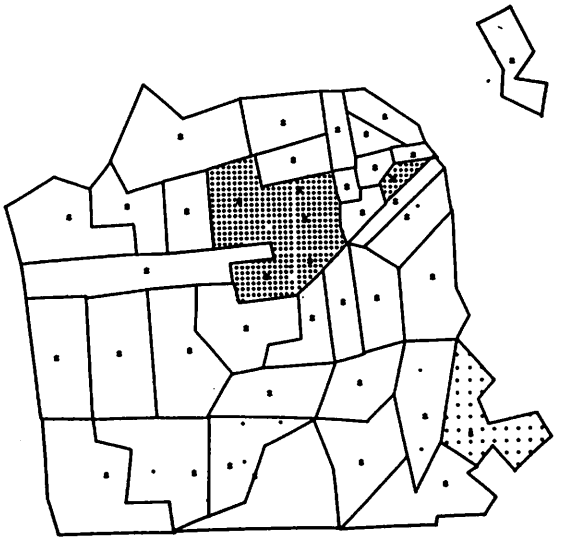


Fig. 11 Combining Sectors. Combining sectors removes common boundaries and produces uniform shading in the unit of combined sectors.

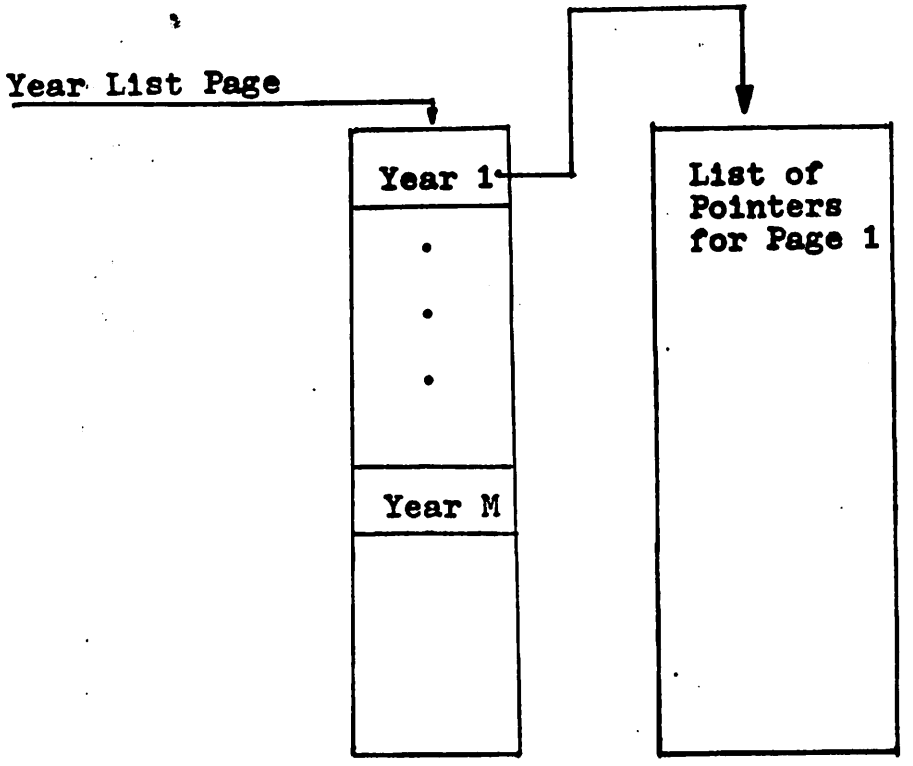


Fig. 12 Directory Paging Structure

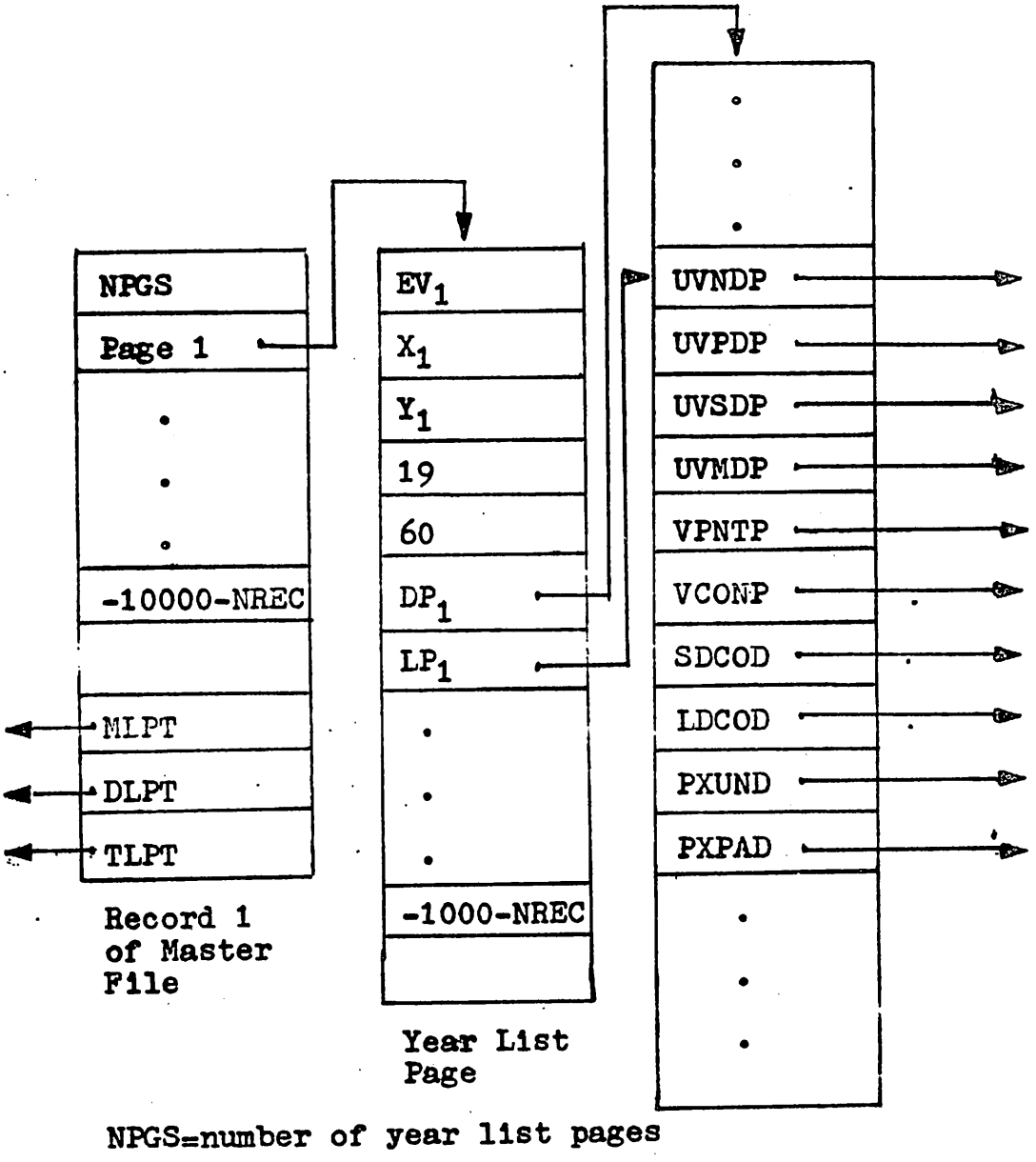
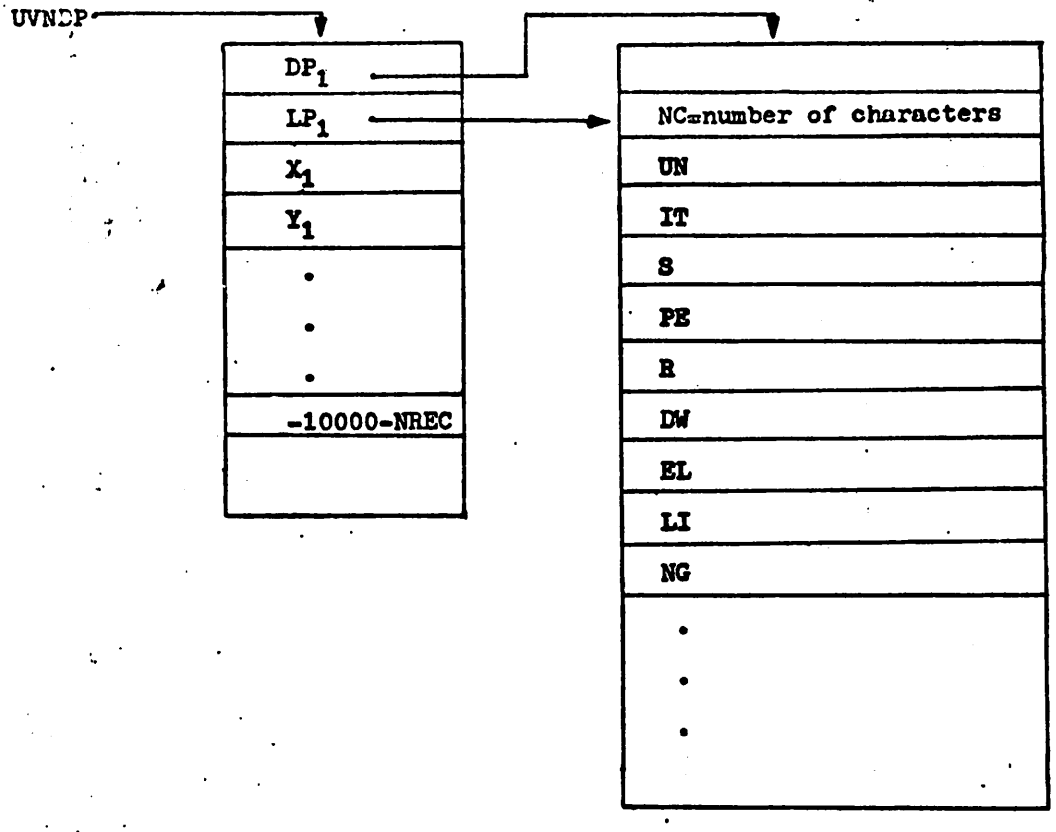


Fig. 13 Detailed Directory Paging Structure



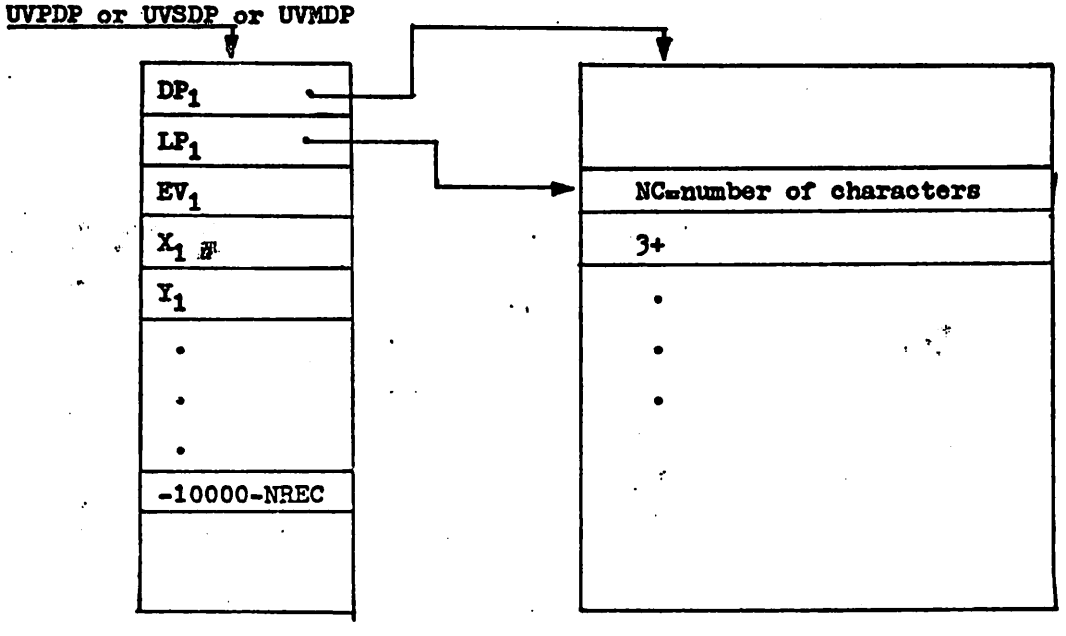


Fig. 14 The Data Structure Necessary to Construct a Page of Basic Variables

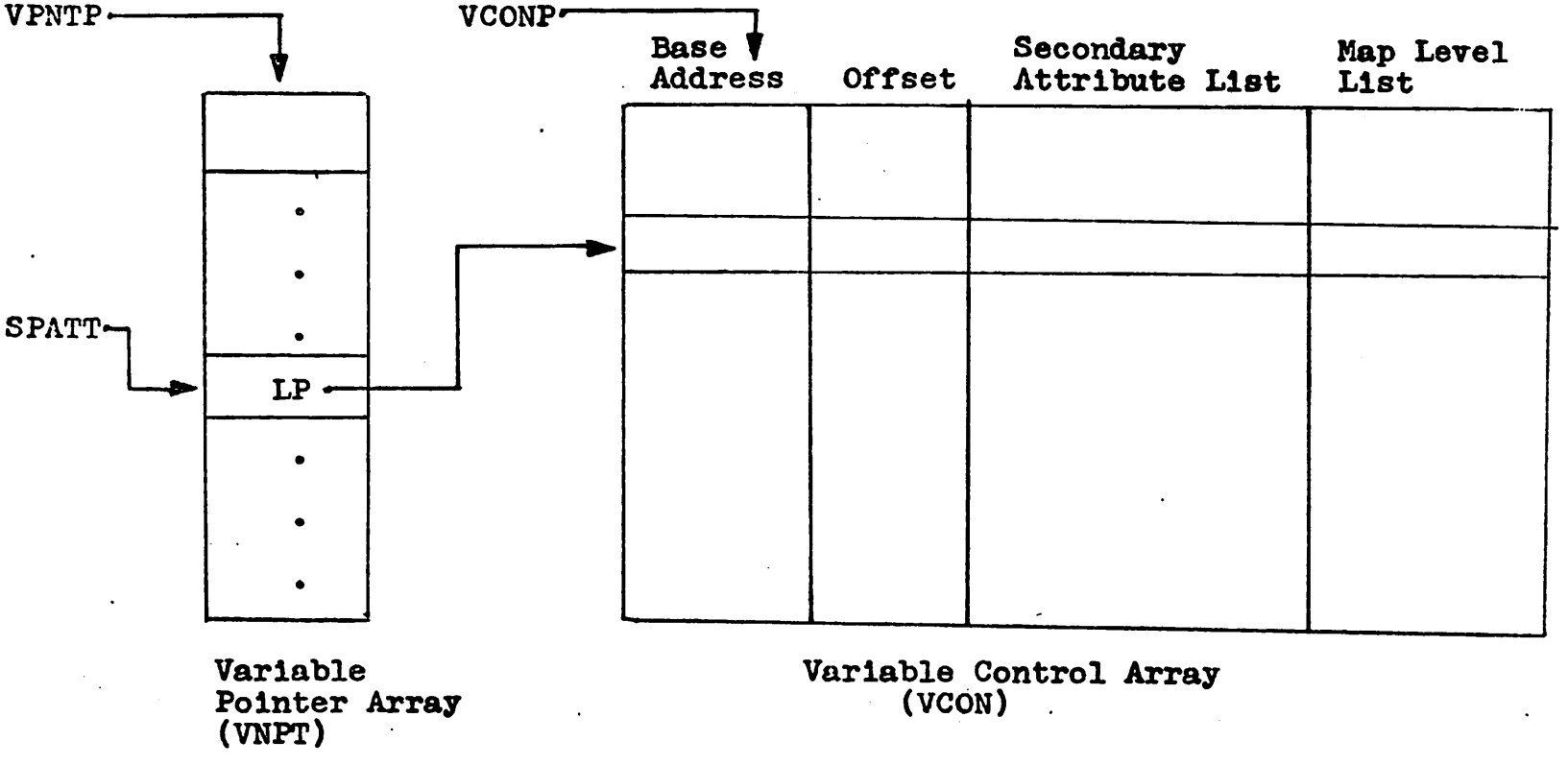
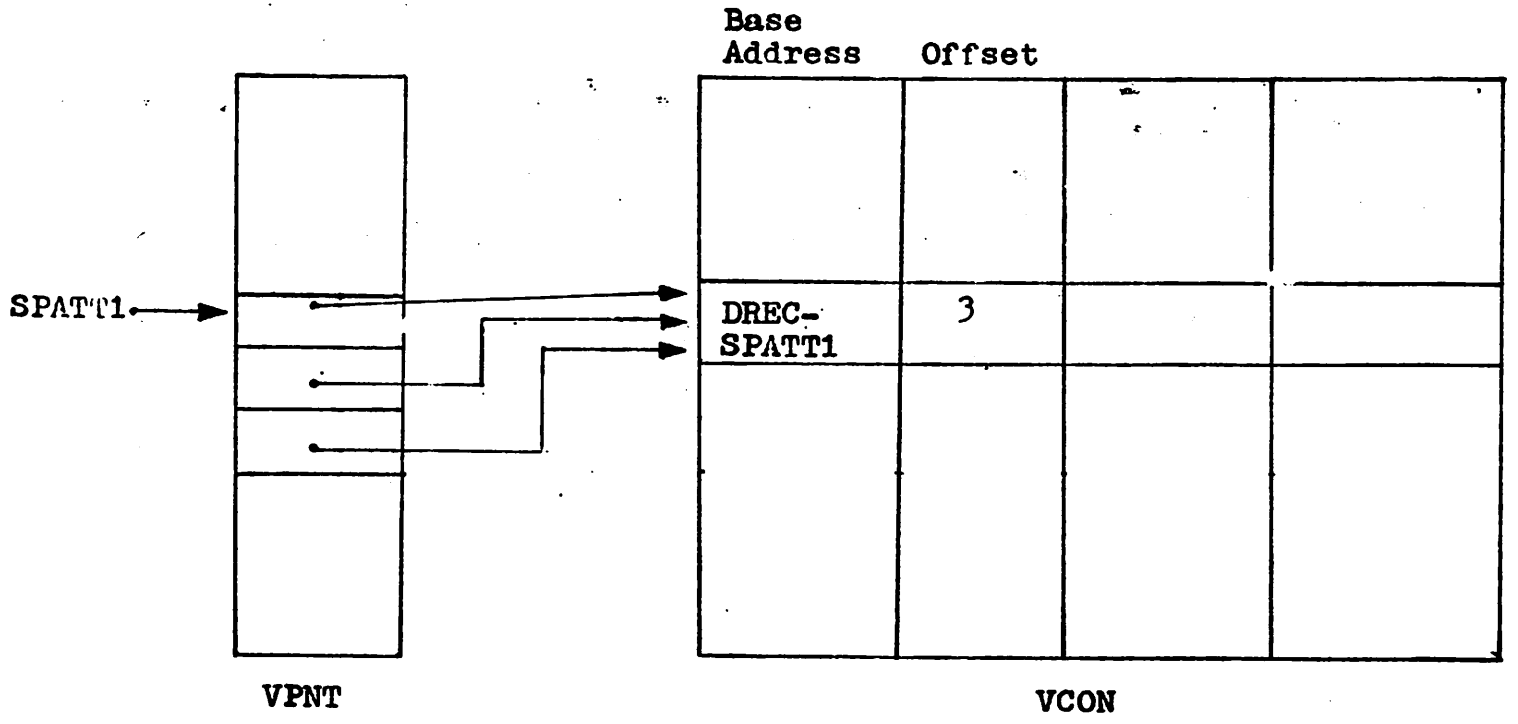


Fig. 15 Structure to Compute Basic Variable Disk Record Address



An urban variable stored in BUDS with 3 primary attributes whose first data record is stored at DREC in some basic variable data file.

Fig. 16 Example of Disc Record Calculation

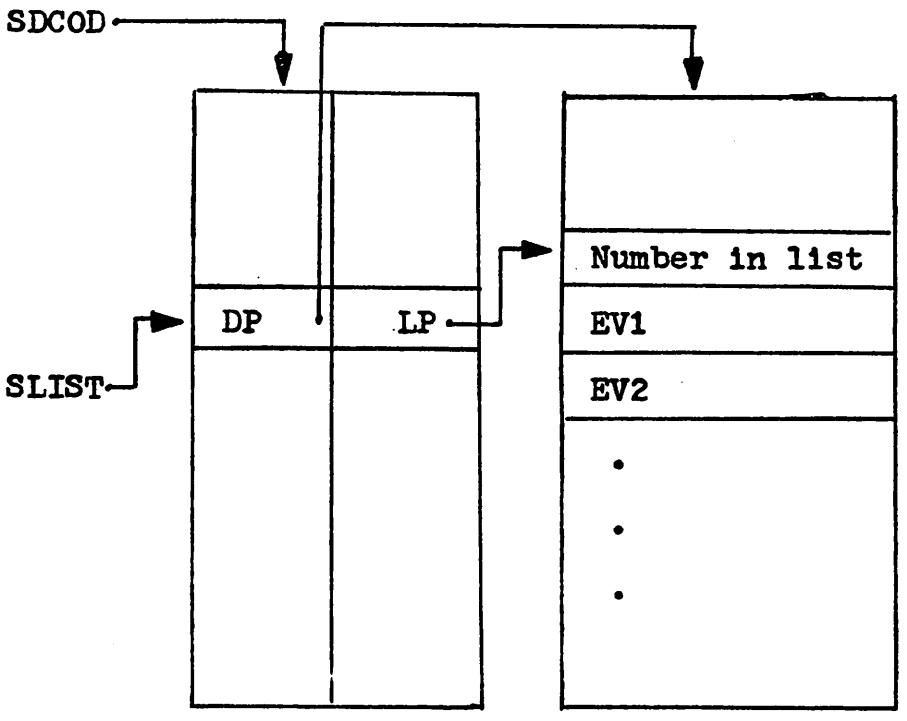
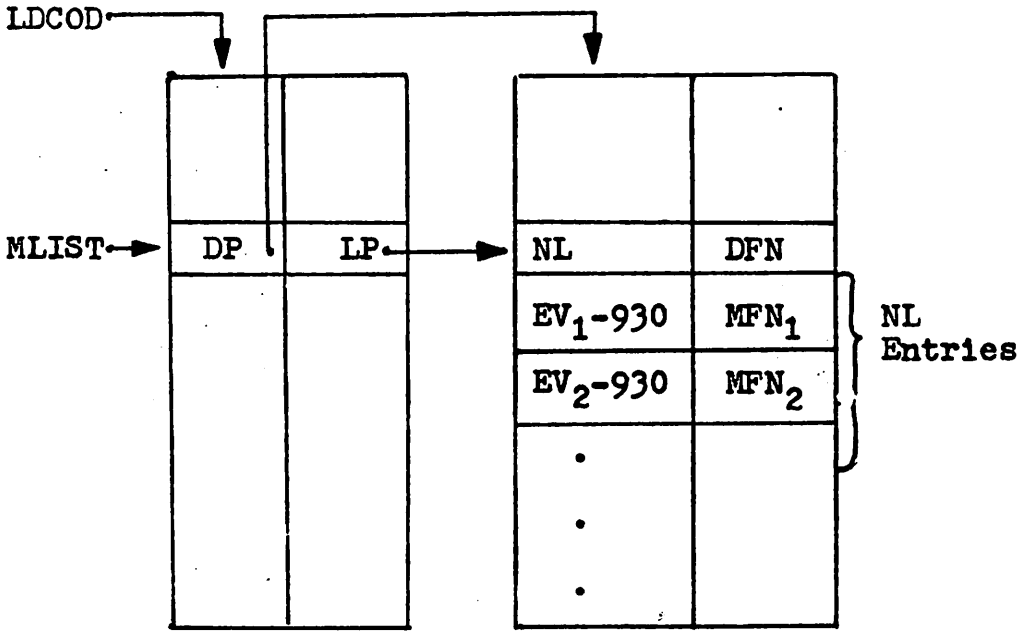


Fig. 17 Secondary Attribute List Retrieval Structure



DFN=data file number
MFN=map file number
NL =number of map levels

This structure is used to determine members of the map list and to decode the selected level into a map file number.

Fig. 18 Map Lists and Decoding Structure

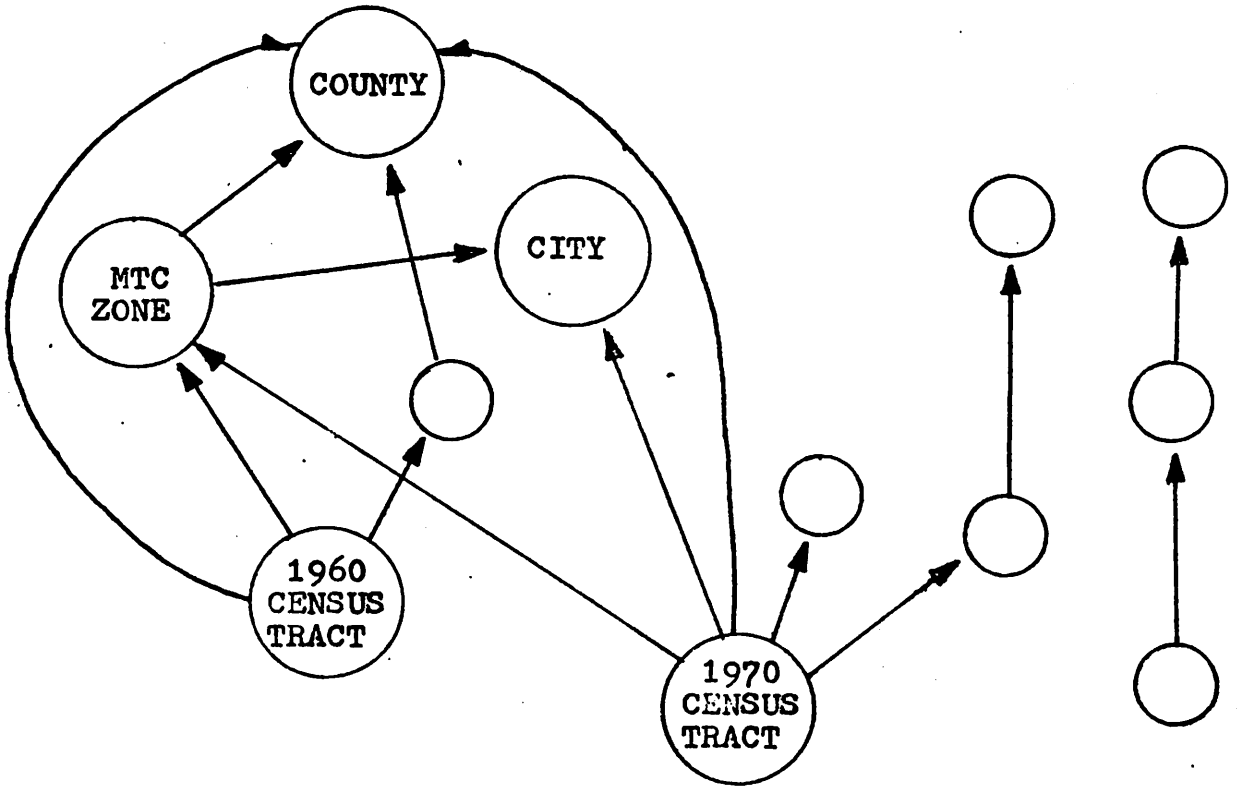
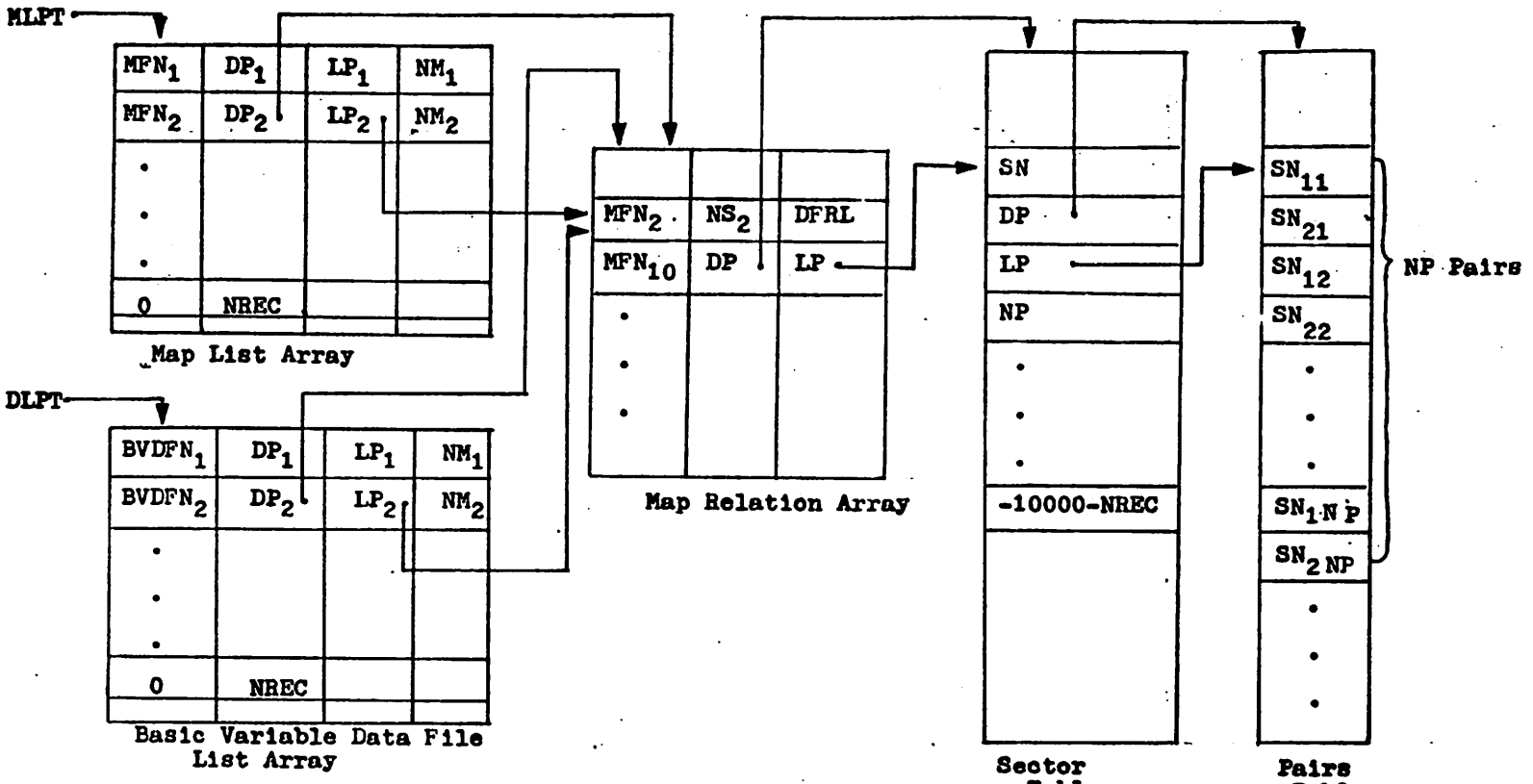


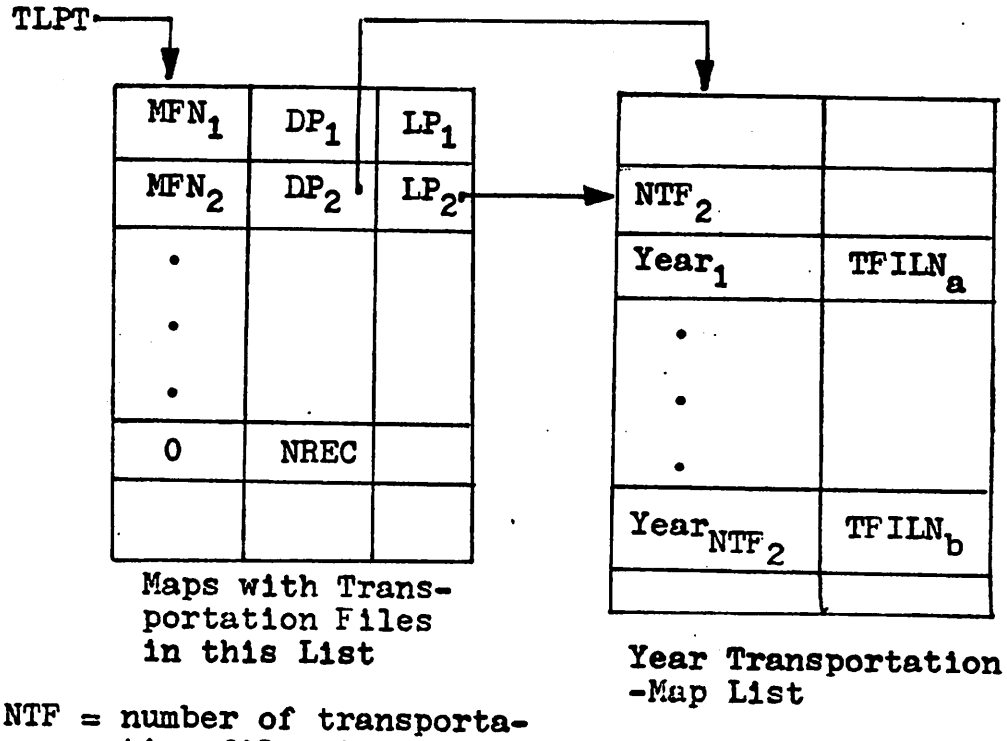
Fig. 19 The Map-Basic Variable Hierarchy Graph



BVDFN = basic variable data file number
MFN = map file number
DP = disk record pointer
LP = array line pointer
NM = number of maps
NS = number of sectors in this map
DFRL = the basic variable data file record length
NREC = the disc record continuing this list

SN = sector number
NP = number of pairs

Fig. 20 Implementation of the Map - Basic Variable Data Hierarchy

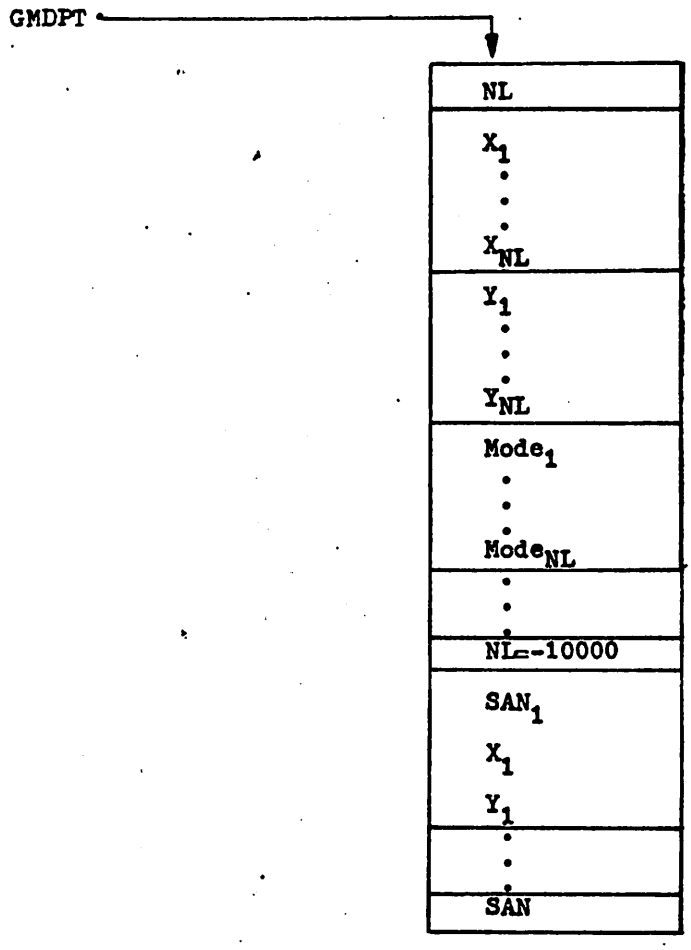


NTF = number of transportation files in this list

Year = EBCDIC character code for the year

TFILN = transportation map file number

Fig. 21 Structure to Match Transportation Maps to the Display Maps

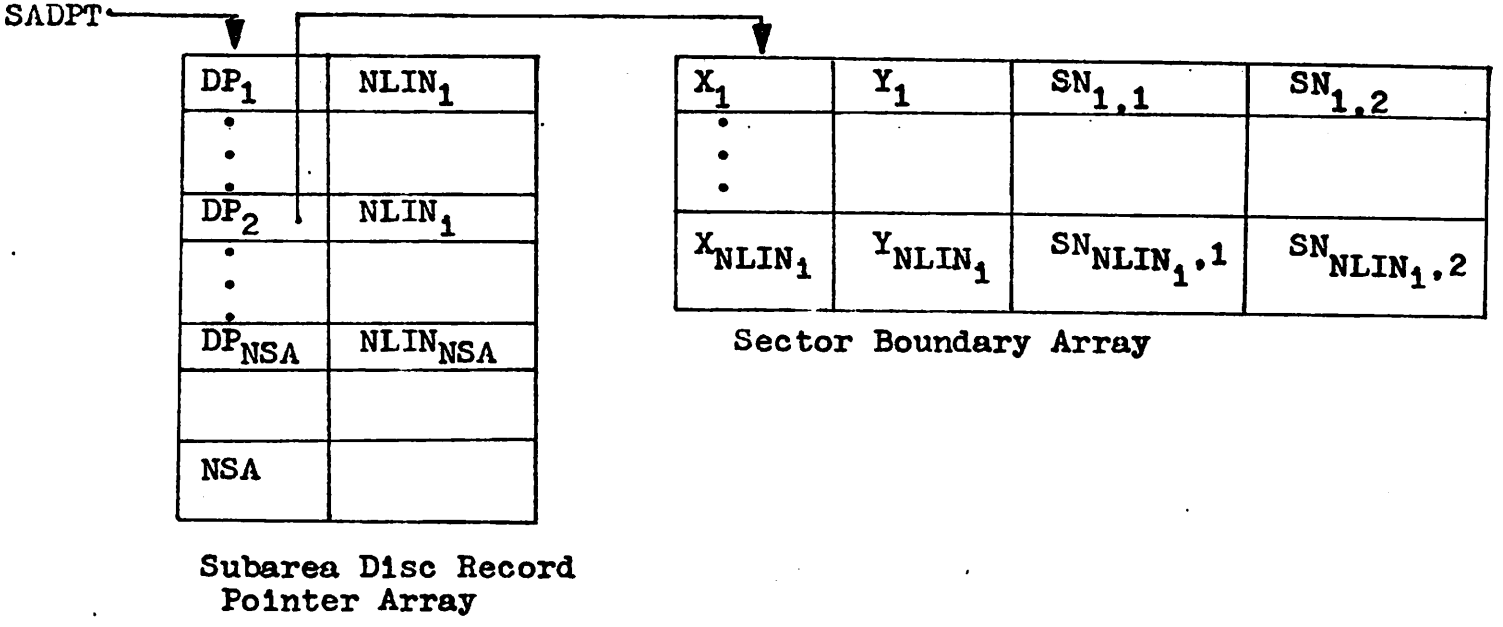


NL = {number of lines if positive
-10000-NREC if negative

$Mode_2$ =
- >5 ignore instruction
- 5 draw absolute point at X1,Y1
- 4 draw absolute line to (X1,Y1) from current beam position
- 3 draw relative point to (X1,Y1) from current beam position
- 2 draw bright relative line to (X1,Y1) from current beam position
- 1 draw relative line to (X1,Y1) from current beam position
- 0 ignore instruction
- -1,-2,-3 draw relative blank line to (X1,Y1) from current beam position
- -4,-5 draw absolute blank line to (X1,Y1) from current beam position
- <-5 ignore instruction

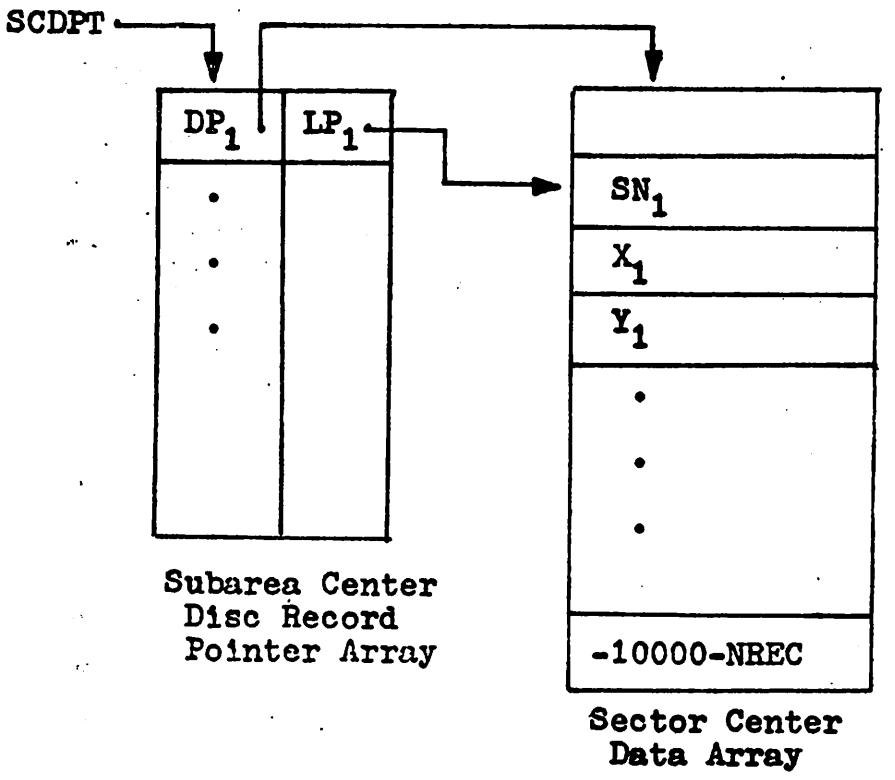
SAN = { subarea number if positive
-10000-NREC if negative

Fig. 22 The Global Map Data Structure Implementation



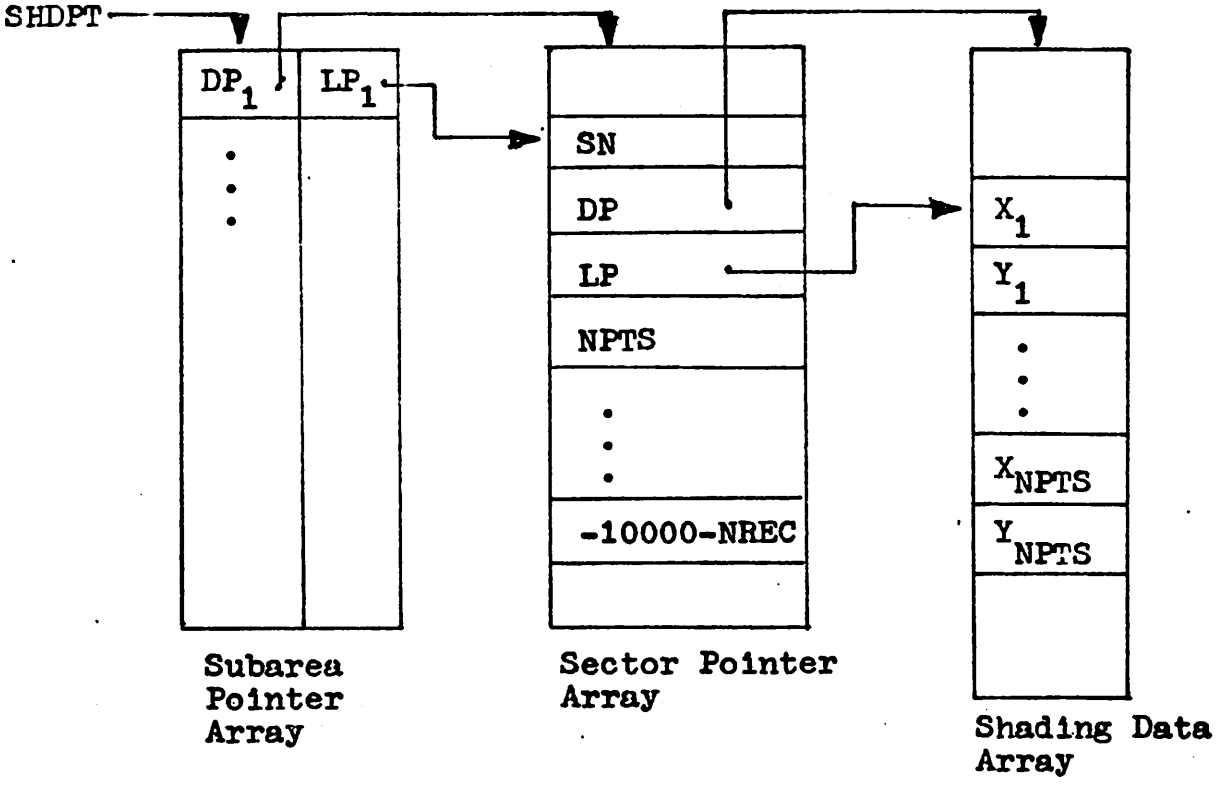
NLIN = number of entries in the array pointed to by DP
NSA = number of subareas in this map
SN = sector number

Fig. 23 The Subarea Sector Boundary Data Structure



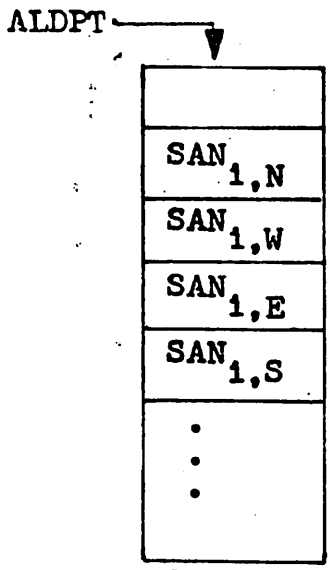
SN = sector number

Fig. 24 The Subarea Sector Center Data Structure

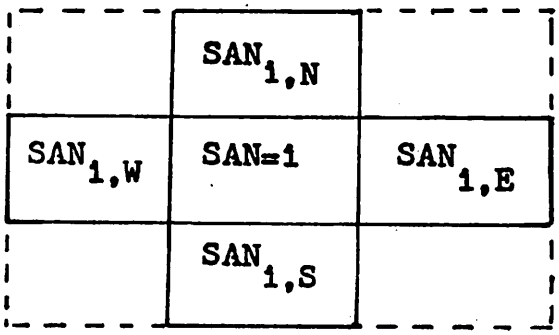


NPTS = the number of sector boundary line segments

Fig. 25 The Subarea Sector Shading Data Structure



Subarea Alignment Data Array

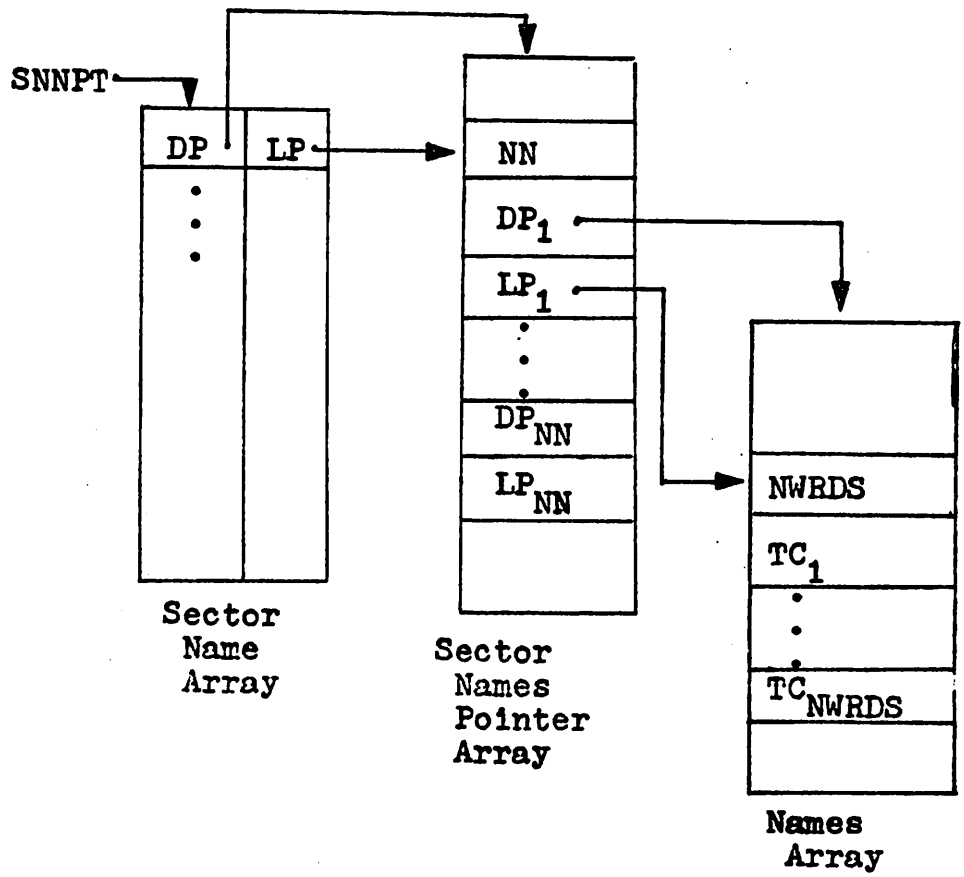


Subarea Location as Indicated by the Subarea Alignment Data

N indicates north position or above
W indicates west position or left
E indicates east position or right
S indicates south position or below

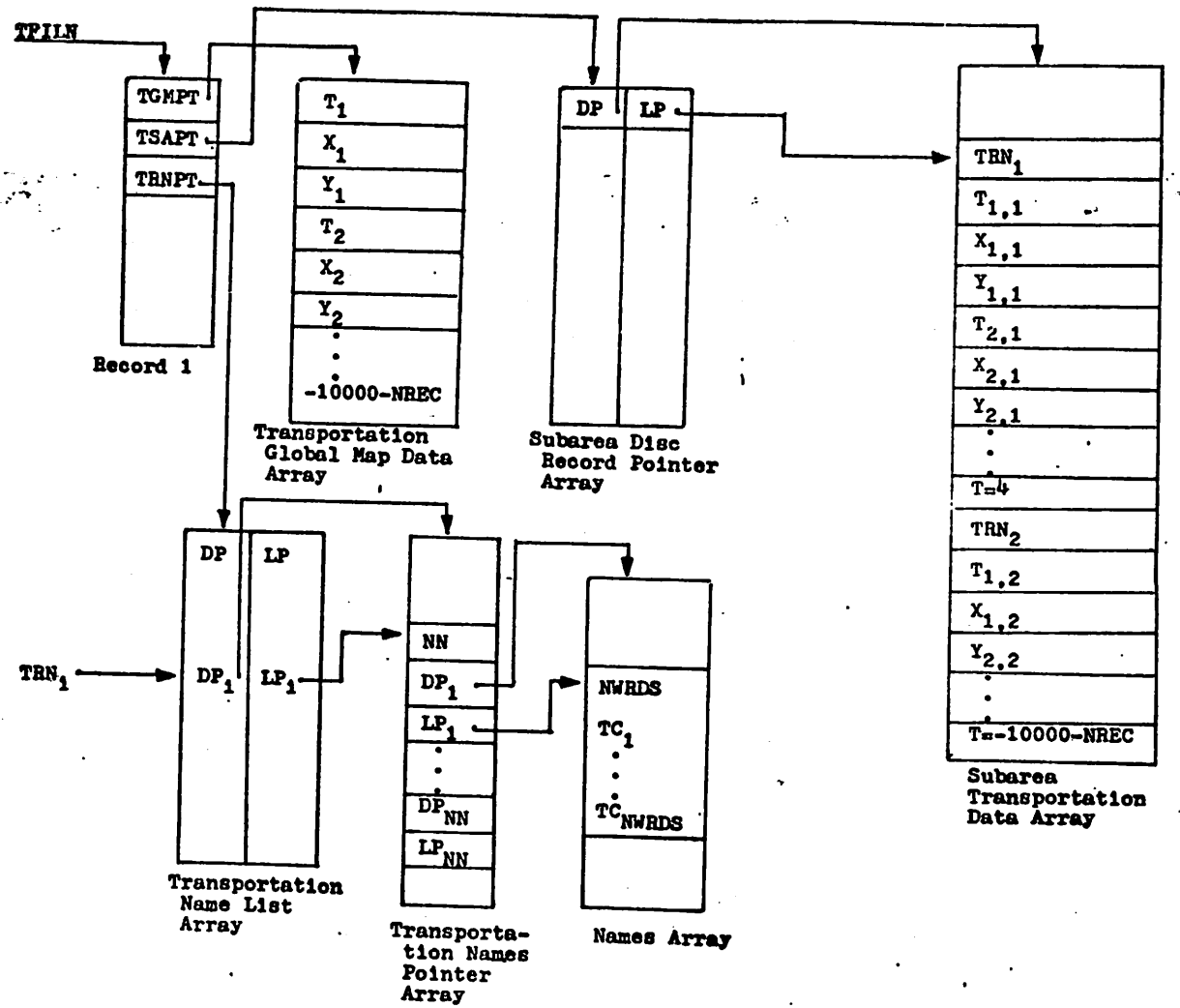
$SAN_{i,P}$ = the subarea number of the subarea in position p the subarea with subarea number i.

Fig. 26 The Subarea Alignment Data Structure



NN = number of names associated with this sector
NWRDS = the number of words necessary for this name
TC = two EBCDIC characters in this cell

Fig 27 The Sector Names Data Structure



T = { 1 Draw blank line; 2 Draw line; 3 Draw point; 4 Start new entity; -10000-NREC Logical record marker }

NN = number of names associated with this facility
NWRDS = the number of words necessary for this name
TC = two EBCDIC characters in this cell
TRN = transportation facility number

Fig. 28 The Transportation File Structure

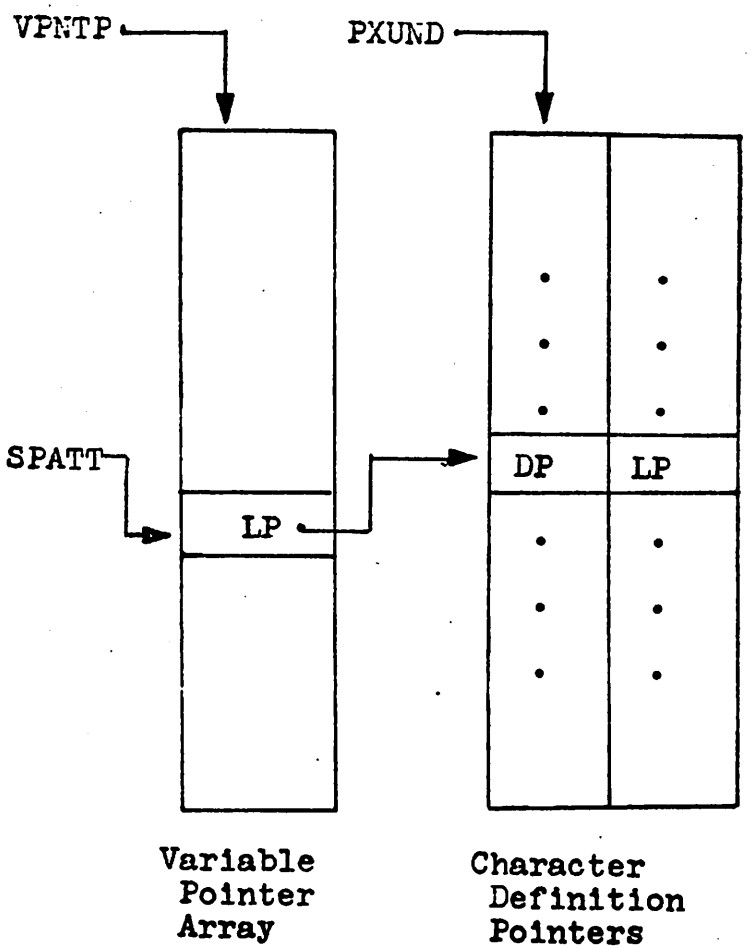


Fig. 29a Structure to Access Urban Name Character Definition

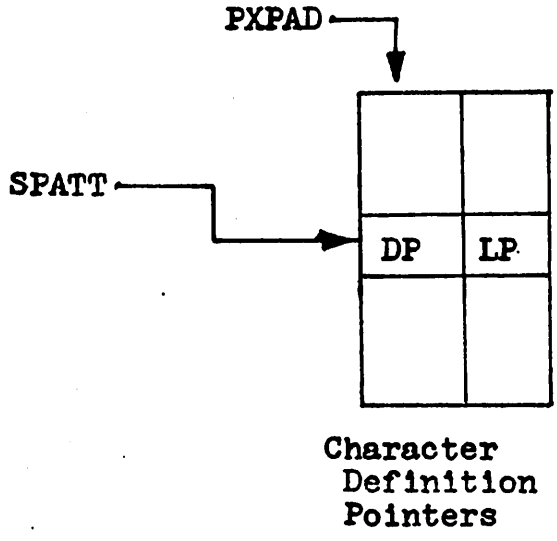


Fig. 29b Structure to Access Primary Attribute Character Definition

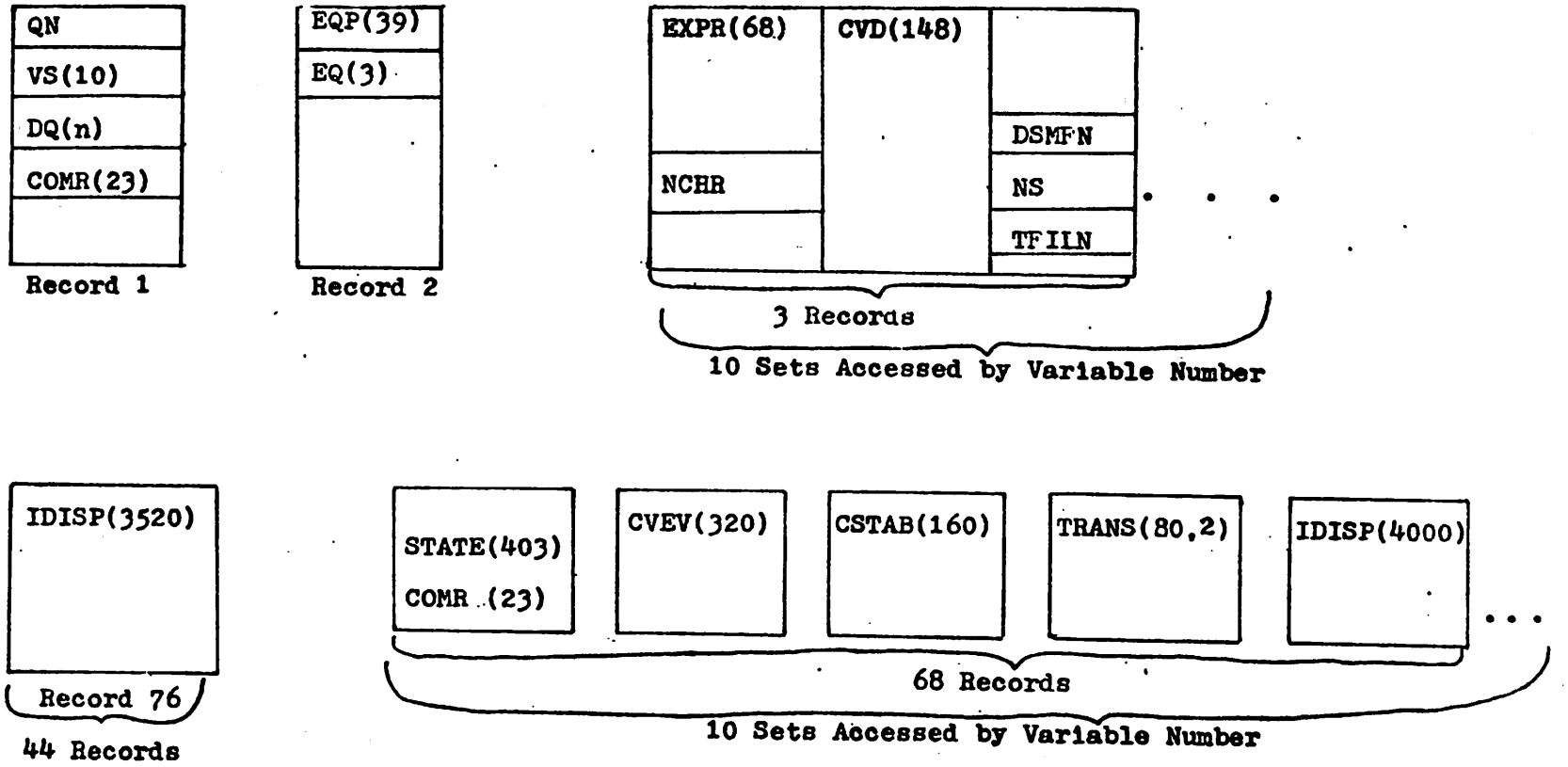


Fig. 30 Structure of PICDA

APPENDIX

A.1. Inputing Numerics for Program Control

This section is concerned with the inputing of integers and real numbers for program control. Entering numerics in the variable expression is discussed in section A.3.

A.1.1. Integers

Some portions of program control expect positive integers while others will accept either positive or negative integers. In what follows whenever only positive numbers are expected an attempt to input a sign will be ignored.

Numeric definition is accomplished via the alphanumeric keyboard. If the user wishes to input a negative integer, minus must be the first character entered followed by appropriate decimal digits. Aside from the sign only cleared digits are accepted. Backspace removes the last character that was entered if there was one entered. While END^{C} terminates the definition.

A.1.2. Reals

There are two types of reals F-format and E-format. F-format numerics are entered only when the MAGNIFY function is used (the format is F5.3). Only decimal characters are accepted. Backspace removes the digit last entered and when there are no more digits the original magnification value appears. Cancel^{C} will will remove all of the entered characters and display the original magnification factor. END^{C} terminates the definition.

E-format numerics are entered to change parameters in a Low-High shading definition. A syntactically correct E-format numeric

consists of a sign (+ or -), followed by one to nine decimal digits with a decimal point anywhere among the digits, followed by the character E, followed by a sign (+ or -), followed by two decimal digits, followed by a blank. END^{C} terminates the definition only if the numeric is syntactically correct otherwise a bleep is sounded and definition continues. A cursor is placed under the character to be edited in an overwrite mode. The space bar moves the cursor one character to the right while backspace moves the cursor one character to the left.

A.2. Computing the Normalization Constant.

Once the variable expression has been computed the values for each sector on the screen is loaded into an array called VAR. These values are then searched for the greatest density variable, GDV, which is the maximum non-negative value and VROFF, the variable offset, or the least non-positive value. Default values for either is zero. Using the value GDV-VROFF (so that the expression value of the sector with the least non-positive value is normalized to zero) the density of shading dots for the sector corresponding to this value is set to 10 raster units per dot. Shading is computed and displayed. If the allocated display memory (4K) is exhausted before all of the shading is displayed, the density of dots is reduced by 15% and shading is recomputed. In the implementation this is accomplished by increasing GDVFC by .15. GDVFC is incremented in this manner until the shading for all sectors on the screen is accomplished without memory overflow.

When shading is computed for a same normalization constant variable GDVFC for the previous variable is used along with VROFF thus insuring the same normalization constant.

A.3. The Variable Expression Grammar

The grammar of the variable expression is the same as that for fortran expressions with minor exceptions. Before giving the grammar some of these differences are discussed.

The expression is limited to 68 characters and it is impossible to insert spaces (blanks) into the expression. Only the unary functions listed below were implemented since these were already in the system library and not enough experience has been gained to determine which new functions should be implemented. However, the task of adding new functions is simple. Unary functions are similar to unary minus, therefore, implementation makes parentheses unnecessary for simple arguments. Thus LN(A) ≡ LNA and LNA + B = LN(A) + B. Also LNLNA ≡ LN(LN(A)). However, either form is acceptable.

Since interest is in recognition and not the properties of the grammar a recognition procedure as implemented in the expression check routine is given.

The procedure is as follows:

Let LP be the left parenthesis counter that is everytime a left parenthesis is encountered in the variable expression LP is incremented by one.

Primitives

Let the variable expression consist of the following primitives.

a - Let FN be the unary function indicator (excluding unary minus). Therefore the occurrence of

AS ≡ absolute value

SN ≡ sine

XP ≡ exponential

LN ≡ natural logarithm

RT ≡ square root

HT ≡ hyperbolic tangent

CS ≡ cosine

in the variable expression is denoted by FN.

b - Let B04 denote the occurence of one of the following binary operators **, *, \ + .

c - Let OP denote the occurrence of a binary operator, B04, or minus (-) both unary and binary.

d - Let Basic Variable denote the occurrence of a basic variable in the variable expression. In the variable expression the letters A, B, ..., J correspond to basic variables. However, the allowable letters thus the length of this list is governed by the number of basic variables in the created variable. For example if the number of basic variables in the created variable is four only the letters A, B, C and D are allowable in the variable expression

e - Let Constant denote the occurrence of a constant in the variable expression. Three types of constants are permissible,

integer, F-format and E-format. Integers may have from one to five decimal digits. F-format constants may have from one to nine decimal digits and a decimal point which can appear anywhere. E-format constants must consist of an F-format constant followed by the letter E followed by a sign (+ or -) followed by two decimal digits.

f - Let (denote a left parenthesis and) denote a right parenthesis.

Procedure

1 - There must be at least one primitive in the variable expression else go to ERROR.

2 - Assume a right parenthesis was recognized and set LP = 0.

3 - If there are no more primitives go to END else set pointer to next primitive.

4 - If the preceding primitive is not (go to 5, else; if this primitive is B04 or) go to ERROR else; set LP = LP + 1 and go to 3.

5 - If the preceding primitive was not a Constant or a Basic Variable go to 6 else; if this primitive is) or OP go to 3 else; go to ERROR.

6 - If the preceding primitive was not FN go to 7 else; if this primitive is B04 or) go to ERROR else; go to 3.

7 - If the preceding primitive was not) go to 8 else, if LP = 0 or this primitive is not OP or) go to ERROR else; set LP = LP - 1 and go to 3.

8 - If the preceding primitive was not OP or - go to ERROR else; if this primitive is OP or) go to ERROR else; go to 3.

END - If the preceding primitive was FN or OP or if LP ≠ 0 go to ERROR else; the variable expression is syntactically correct.

ERROR - The variable expression is snytactically incorrect.

A.4. Converting Primary Attribute Entity Values to Character Definitions.

When basic variables are selected from the directory their character definitions must be retrieved in order to enter them in created variable definitions for display on the CRT and for printing on the typewriter. Fig. 29 illustrates the data structure to accomplish this. The sections in Chapter 2 expecially 2.2.3 and the first six sections of Chapter 3 should be read before attempting to read this section.

Fig. 29a illustrates the data structure used to access urban name character definitions. The variable pointer array was discussed in section 3.2.2 and is illustrated in Fig. 15. SPATT the selected primary attribute index is used as a pointer into this array to access a line pointer. This line pointer is used as an index into an array pointed to by the primary attribute entity value to urban name disc record pointer. This yields pointers to the character definition of the urban name associated with SPATT (refer to Fig. 14).

Fig. 29b illustrates the data structure used to access urban name character definitions. The variable pointer array was discussed in section 3.2.2 and is illustrated in Fig. 15. SPATT the selected primary attribute index is used as a pointer into

this array to access a line pointer. This line pointer is used as an index into an array pointed to by the primary attribute entity value to urban name disc record pointer. This yields pointers to the character definition of the urban name associated with SPATT (refer to Fig. 14).

Fig. 29b illustrates the data structure necessary to access the primary attribute character definitions. Contrary to the above SPATT is used as a line pointer into an array pointed to by PXPAD, the primary attribute entity value to primary attribute name disc record pointer. This yields pointers to the character definition of the primary attribute associated with SPATT (refer to Fig. 14).

A.5. The Picture Data File

The picture data file, PICDA, contains all of the semi-permanent computed data. This includes created variable definitions, created variable expression results and CRT picture definitions as well as other data. Logical records are 80 words in length and the description of the file will be in terms of logical records whenever possible. Figure 30 gives the overall structure of PICDA.

The first record contains QN, VS(10), DQ(11) and COMR(23). QN, the acronym for queue number, is an integer constant and is used as a transfer indicator. It has the value -1 while transfering from DDQEP or VARDP or if program operation was terminated in DQEP. If program operation was terminated in VARDP, QN will equal the queue number of the variable being displayed and hence is used as

an index to access the appropriate data necessary to continue program operation at the point of termination (i.e. reestablish the state of BUDS) this is discussed below. While transfering from DQEP to VARDP it has the value 11.

VS, the acronym for variable status array, is an integer array accessed by queue number and is ten words in length. The bit configuration of each word in this array gives the status of the created variable associated with this queue number. The following table gives bit configuration meanings. The bit is set to one if the condition is met.

Bit	Condition
15	variable expression calculated
14	map constructed
13	shading constructed
12	variable is a "same normalization contant" type
11	variable is a Low-High type
10	sectors combined

Table 1: Variable status bit configuration.

DQ, the acronym for display queue, is an integer array of length eleven. DQ(11) contains the number of created variables on the display queue (NVDQ). The first ten cells of DQ contain the integers one through ten not necessarily in order. Each of these cells corresponds to a queue number and a variable number. The queue numbers, QN, is an index into DQ(10) whereas the variable number, VN, is contained in that cell (i.e. DQ(QN) = VN). There

is a great deal of data associated with each created variable. This structure was adopted so that enormous quantities of data need not be shifted around by a reordering of created variables on the display queue. For example, if the created variable with QN = 1 is deleted all that has to be done is to move the variable numbers in DQ.

COMAR is an integer array of 23 cells. It contains vital graphics communication information.

EQP, the edit queue entity pointer array, is an integer array with 39 cells. The graphics software requires every defined entity to have a entity pointer. EQP is used by DQEP so that three created variables may be displayed at once. Thirteen pointers are assigned to each screen location. The first thirteen to screen location one and so forth. The first two cells are used for created variable header entities. The third cell is used as the variable expression entity pointer. The remaining ten entity pointers are used as basic variable entity pointers.

EQ, the edit queue, is an integer array of three cells. It contains the variable numbers of the variables displayed for editing in screen locations one to three. If a variable is not in a screen location the corresponding cell of EQ will be zero.

EXPR, the variable expression definition, is an integer array of 68 cells. It contains the variable expression definition one character per cell and right justified. NCHR is the number of characters or cells in the variable expression.

CVD, the created variable definition, is an integer array of

148 cells. The first cell contains the number of basic variables in the definition. The next seven cells contain header information. The remaining 140 cells are used for basic variable definitions; fourteen cells per basic variable. DSMFN is the display map file number and NS is the number of sectors in this map. TFILN is the transportation map file number for the display map.

EXPR and CVD are stored in PICDA by variable number.

IDISP(3520) is an integer array with 3520 cells. This array contains the master display list and display area for DQEP which was previously referred to as the display memory. The display list contains entity pointers of entities which are currently on display. Note that an entity may be defined but not on display. Also an entity is brightened by placing its entity pointer in the display list three times. The display area contains all entity definitions. That is the 2250 graphics instructions necessary to create an entity on the CRT.

STATE, the VARDP state of a created variable, is an integer array of 403 cells. It contains most of state information of the created variables. COMAR is the vital graphics communications information for this variable in VARDP.

CVEV, the computed variable expression sector values, is a real array of 640 cells or 320 words. Presently the maximum number of sectors allowable in any map is 320. When the variable expression is computed it is computed for every sector in the display map and these values are stored in CVEV.

CSTAB, the combined sectors table, is an integer array of 160

cells contains the information generated by the COMBINE function to combine sectors.

TRANS, the transportation facility entity pointer and transportation facility number array, is an integer array with 160 cells. Transportation facility numbers are analogous to sector numbers. They are used to identify transportation facilities.

IDISP(4000) contains the display list and display information for the created variable corresponding to the variable number associated with IDISP in VARDP.

BIBLIOGRAPHY

1. Forrester, J. W. Urban Dynamics, MIT Press, Cambridge, Mass (1969).
2. The Bay Area Simulation Model (BASS), the Staff of the Center for Real Estate and Urban Economics, University of California, Berkeley (1967).
3. SICCAPUS Bulletin, Vol. 2, No. 3, July (1968) ACM, p. 23
4. Digital Scientific Meta 4 Series 16 Computer System Reference Manual, Digital Scientific Corporation, San Diego, Calif. (1970).
5. Birch, David, L., "Toward a Theory of Urban Growth," AIP Journal, March 1971 pp. 78, 87.
6. Hoover, Edgar M. and Vernon, Raymond, Anatomy of a Metropolis Harvard University Press Cambridge, Mass. (1959).
7. The Data Text System: A Computer Language for Social Sciences Research, Harvard University, Cambridge, Mass. (1969).
8. Hie, Norman H. et al. SPSS: Statistical Package for the Social Sciences, University of Chicago, Chicago, Ill (1968).
9. Miller, James R, "Datanal: An Interpretive Language for On-Line Analysis of Empirical Data," Working paper No. 275-267, Sloan School of Management, MIT (August 1967).
10. Rogers Andre, "Theories of Intra-Urban Spatial Structure: A Dissenting View" Land Economics 43 February 1967, p 108.
11. Groat, M. F., Senior City Planner, City of San Francisco, Calif. Personal Communication August 1973.
12. Cristiani E. I., Evey R. J., Goldman, R. E., Mantey P. E., "An Interactive System for Aiding Evaluation of Local Government Policies," IEEE Trans. Systems, Man and Cybernetics. Vol. SMC-3, No. 2, March 1973, pp. 141-146.

13. Mantey, P. E., Bennett, J. L., Carlson, E. E., "Information for Problem Solving: The Development of an Interactive Geographic Information System," Forthcoming IBM San Jose Research Laboratory, San Jose, Calif. 1973.

14. Parker, James L., "The Natural Information Systems Project An Overview," Internal paper, Dept. of Computer Science, Univ. of British Columbia, Vancouver, British, Columbia, August 8, 1971.

15. Parker, James L., "Information Retrieval with Large-scale Geographic Data Bases," Dept. of Computer Sciences, Univ. of British Columbia, Vancouver, B.C., Report No. 1, June 1971.

16. Parker, James L., "A Graphics and Information Retrieval Supervisor for Simulators," Dept. of Computer Science, Univ. of British Columbia, Vancouver, B. C., Report No. 2, September 1971.

17. Schumacher, Betsy, "Urban Cogo - A geographics - based land information system," FJCC 1971, pp. 619, 630.

18. Kadanoff, L. P., Voss, J. R. and Bouknight, W. J., "Display of Urban Growth Patterns," IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-7, No. 3, May 1971, pp. 448, 458.

19. Loomis, R. G. and Lorenzo, J. J.m "Experiments in Mapping with a Geo Space Plotter," Urban and Regional Information Systems for Social Problems, Garden City, N. Y., John E. Pickert Editor, 1967, p. 219.

20. Dudnik, Elliot E., Synagraphic Mapping System - SYMAP manual, Department of Archetecture and Art, University of Illinois, Chicago Circle.

21. Codasyl Systems Committee, Feature Analysis of Generalized Data Base Management Systems Technical Report, ACM, NY, NY, May 1971.

22. Hearle, Edward F. R., "Information Systems in State and Local Governments," A. R. of Information Science and Technology, Vol. 5, pp. 325, 349, 1970.

23. Dial, O. E., Urban Information Systems: A Bibliographic Essay, Urban Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., 1968.

24. Licklider, J. C. R., "A picture is worth a thousand words - and it costs..." Spring Joint Computer Conference, AFIPS 1969 pp. 617-621.

25. Lewin, Morton H., "An Introduction to Computer Graphics Terminals," Proceedings of IEEE Sept 1967, pp. 1545-1552.

26. Foley, James D., "Evaluation of Small Computers and Display Controls for Computer Graphics," Compuer Group News, Jan-Feb 1970, pp. 8, 21.

27. Saltz, Warren L., "Survey of Hardware R&D for Computer Display," Computer Design, May 1972, pp. 89, 154.